Final Project for SW Engineering CSC 648-848 Section 01 Summer 2021

DormMates

Milestone 5 • Version 1 • 5 August 2021

Team 01

Team Lead and Github master	Andrei Georgescu	ageorgescu@mail.sfsu.edu
Frontend Lead	Meeka Cayabyab	
Backend & Database Lead	Jonathan McGrath	
Engineer	Alexandre Ruffo	
Engineer	Jimmy Yeung	
Engineer	Sayed Hamid	
Engineer	Ahad Zafar	

History Table

Version	Date	Notes
M5V2	N/A	N/A
M5V1	08/05/2021	Initial submission
M4V2	08/03/2021	Addressed feedback and updated non-functional requirements status
M4V1	07/30/2021	Initial submission
M3V2	07/30/2021	Addressed feedback
M3V1	07/22/2021	Initial submission
M2V2	07/19/2021	Addressed the vertical prototype, functional requirements, and diagrams feedback.
M2V1	07/08/2021	Initial submission
M1V2	07/01/2021	Added database master role to the title page, addressed use cases feedback, addressed functional requirements feedback, and addressed competitive analysis feedback.
M1V1	06/22/2021	Initial submission

Table of Contents

Table of Contents **3**
Product Summary **6**
Final Priority 1 Functions 6
Unique Features 7
URL 8
Milestone Documents **8**
Milestone 1, Version 2 8
Milestone 2, Version 2 45
Milestone 3, Version 2 87
Milestone 3, Horizontal Prototype Feedback 106
Milestone 4, Version 2 111
Product Screenshots **148**
DB Table Screenshots 159
User Table 159
Student Table 159
Landlord Table 159
Listing Table 160
Amenities Table 160
Institution Table 160
Task Management System Screenshots **161**
Screenshot of Sprint in Jira 161
Screenshots of Some High-Level Tasks and Subtasks 161
Screenshot of Detailed View of a High-Level Task 162
Team Member Contributions **163**
Post Analysis 164

Product Summary

DormMates

Final Priority 1 Functions

1.1. Unregistered User
- 1.1.1. An unregistered user can create a new student or landlord account.
- 1.1.2. An unregistered user can view listings near an institution of their choice.
- 1.1.3. An unregistered user can view the Home page.
- 1.1.4. An unregistered user can view the Terms of Service page.
- 1.1.5. An unregistered user can view the FAQ page.
- 1.1.6. An unregistered user can view the Features pages.
- 1.1.7. An unregistered user can view the About page.

1.2. Registered User
- 1.2.1. A registered user should be able to login to with their username and password.
- 1.2.2. A registered user should be able to logout of their account.
- 1.2.3. A registered user should be able to change their username.
- 1.2.4. A registered user should be able to change their email address.
- 1.2.5. A registered user should be able to change their password.

1.3. Students
- 1.3.1. A student user must have a verified edu email.
- 1.3.2. A student user must have a completed profile.
- 1.3.3. A student user shall be able to view the student dashboard.
- 1.3.4. A student user shall be able to view student user profiles.
- 1.3.5. A student user shall be able to search for listings.
- 1.3.6. A student user shall be able to edit their personality.
- 1.3.7. A student user should be able to edit their schedule.
- 1.3.8. A student user should be able to edit their hobbies.
- 1.3.9. A student user shall be able to filter listings by amenities.
- 1.3.10. A student user shall be able to filter listings by distance from university.
- 1.3.11. A student user shall be able to filter roommate selections by personality.
- 1.3.12. A student user shall be able to filter roommate selections by major.
- 1.3.13. A student user shall be able to filter roommate selections by hobbies.
- 1.3.14. A student user shall be able to filter roommate selections by schedule.
- 1.3.15. A student user should be able to view the location of a listing on a map.
- 1.3.16. A student user shall be able to favorite a listing.

1.4. Landlords
   1.4.1. A landlord user shall be able to view the landlord dashboard.
   1.4.2. A landlord user shall be able to view their own profile.
   1.4.3. A landlord user shall be able to post listings.
   1.4.4. A landlord user shall be able to edit their listings.
   1.4.5. A landlord user shall be able to view student profiles.
   1.4.6. A landlord user shall be able to delete their listings.

Unique Features

DormMates targets college students who would like to search for both housing and roommates. Students who sign up on our website will have the ability to look for potential roommates who share the same interests using our roommate filtering system. Students will additionally have the option to filter through housing listings to find their ideal housing. Those who want to list housing can create a landlord account and post a listing as well.

URL

https://dormmates.net

Milestone 1, Version 2

SW Engineering CSC648 Summer 2021

DormMates

Milestone 1 • Version 2 • 22 June 2021

Team 01

Team Lead and Github master	Andrei Georgescu	ageorgescu@mail.sfsu.edu
Frontend Lead	Meeka Cayabyab	
Backend and Database Lead	Jonathan McGrath	
Engineer	Alexandre Ruffo	
Engineer	Jimmy Yeung	
Engineer	Sayed Hamid	
Engineer	Ahad Zafar	

History Table

Version	Date	Notes
M1V2	07/01/2021	Added database master role to the title page, addressed use cases feedback, addressed functional requirements feedback, and addressed competitive analysis feedback.
M1V1	6/22/2021	Initial submission

Table of Contents

Executive Summary 3

Main Use Cases 4

List of Main Data Items and Entities 18

Functional Requirements 20

- User 20

Non-Functional Requirements 23

- Functionality 23
- Security 23
- Privacy 23
- Legal 23
- Performance 24
- System Requirements 24
- Marketing 24
- Content 24
- Scalability 24
- Capability 25
- Look and Feel 25
- Coding Standards 25
- Availability 26
- Cost 26
- Storage 26
- Expected Load 27

Competitive Analysis 28

High Level System Architecture and Technologies 31

Checklist 32

List of Team Contributions 33

Executive Summary

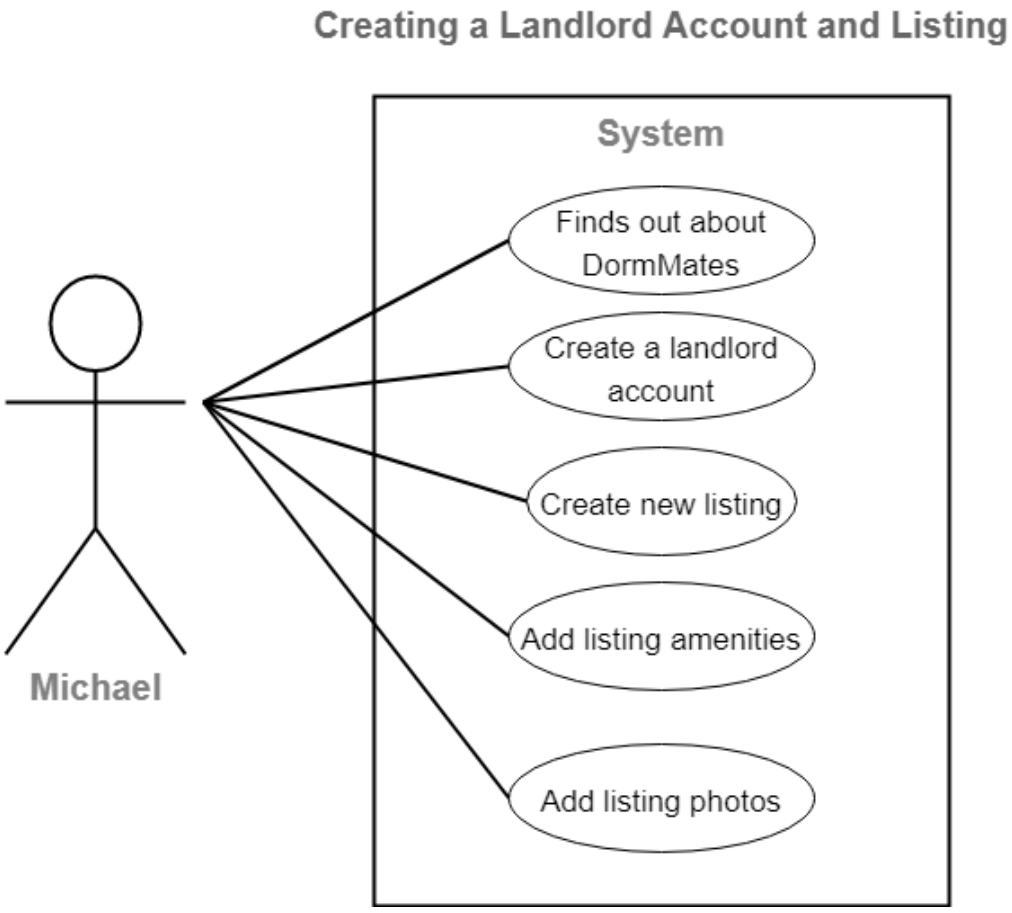
One of the most challenging things to do as a college student is finding the right living situation for you. For example, San Francisco State University only has housing available for less than 12% of registered students. As a third of new incoming freshmen are becoming more concerned with the ever-increasing prices of housing, they must resort to other means to find affordable housing. These students may find themselves utilizing various different websites and tools, making their search for housing inconsistent. Students may use Facebook to find the right group of roommates and Craigslist to find the right apartment. DormMates aims to simplify this entire process as it allows students to find both satisfactory roommates and housing that aligns with their needs in one easy-to-use platform.

Living away from home for the first time can be an overwhelming experience for most college students. The notion of having to live with a random person that you know nothing about, in a location you did not choose can greatly impact a student's emotional and academic success. DormMates helps students get paired with roommates based on their academic major, hobbies, interests, schedules, and lifestyle. In addition to this, DormMates can also be used by landlords to list their housing unit. Landlords can chat with multiple students before deciding on an agreement and they are held to a high standard by our Landlord rating system to ensure students find the best housing option possible.

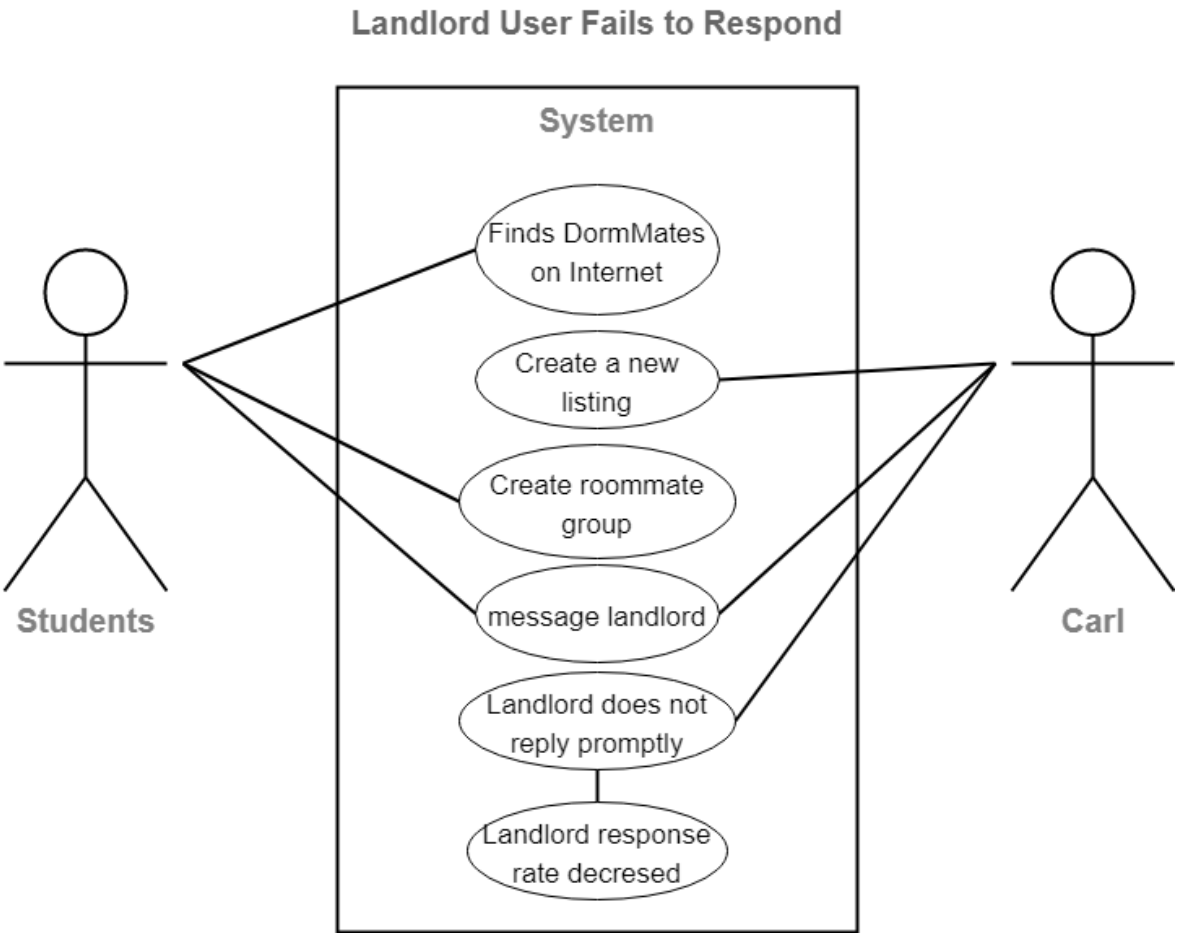
DormMates seeks to empower college students to take their living situation into their own hands by providing them a platform to connect with landlords and other students. We aim to give both students and landlords peace of mind when searching for roommates or listing their units on DormMates through several key features. Firstly, DormMates requires all students to have a valid and verifiable university email address. Next, our roommate matching service will utilize surveys and other techniques to match students with the right roommates for them. Lastly, our landlord rating system will ensure that all landlords provide students with the highest quality of living possible.

Main Use Cases

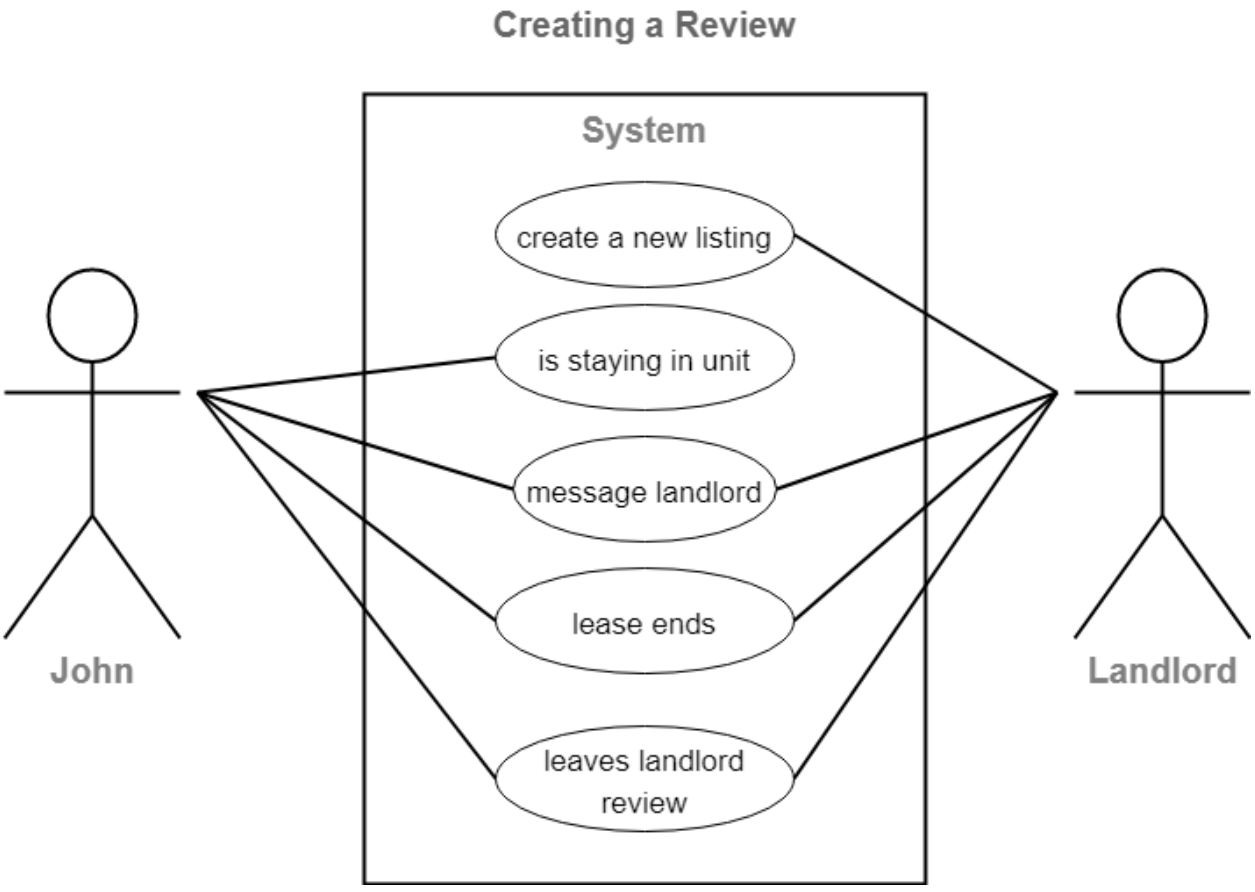
Title:	Creating a Landlord Account and Listing
Actor(s):	Michael
Description:	A landlord, Michael, purchases his first rental unit and is looking for college students to be his tenants. Michael contacts a friend who has been renting his rental unit to students and tells Michael about DormMates. Michael then creates a DormMates Landlord account. He creates a new listing for his unit. Michael also compiles a list of all amenities his unit has to offer and realizes that his unit can comfortably host 4 college students.



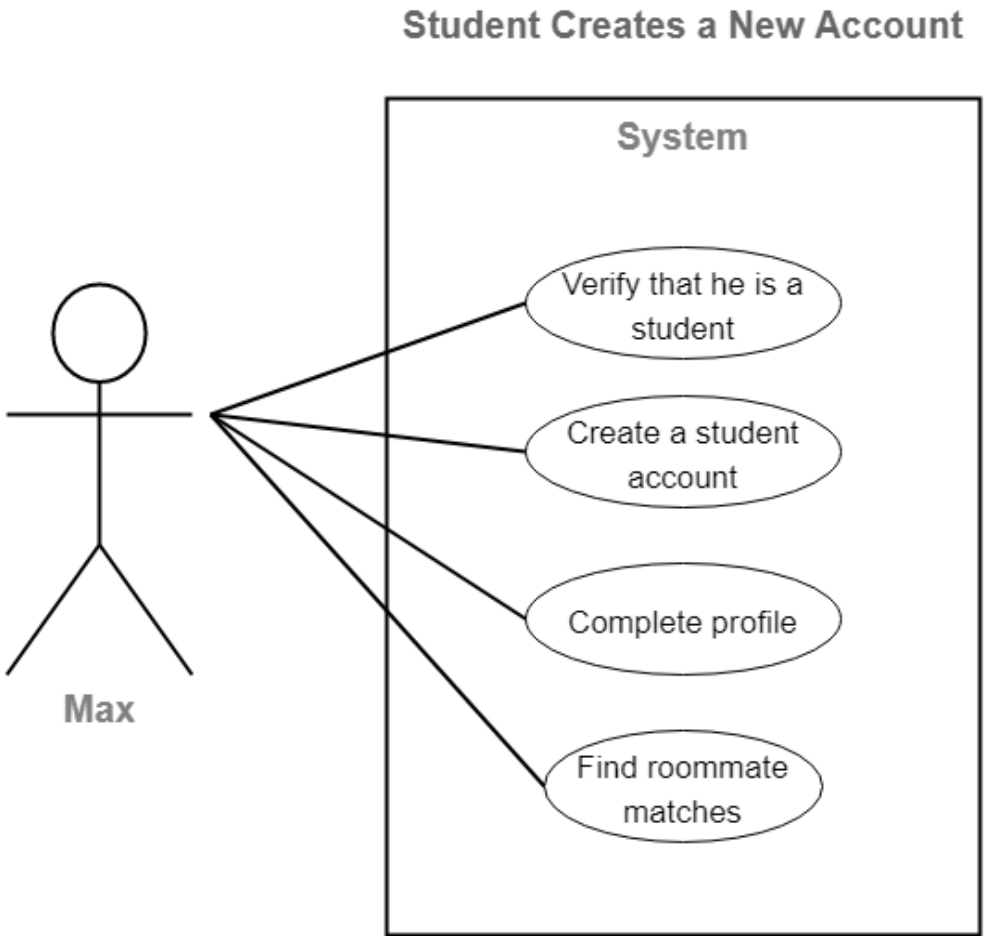
Title:	Landlord User Fails to Respond
Actor(s):	Students and Carl
Description:	A group of students wants to find roommates from the same school and same hobbies as them. After browsing through the internet they find DormMates. They create a student account and search for available housing listings, they find an apartment that suits all of their needs that is owned by Carl, a landlord. The group of students messages Carl through DormMates. When Carl fails to respond to the group within a limited amount of time his response rate rating is decreased.



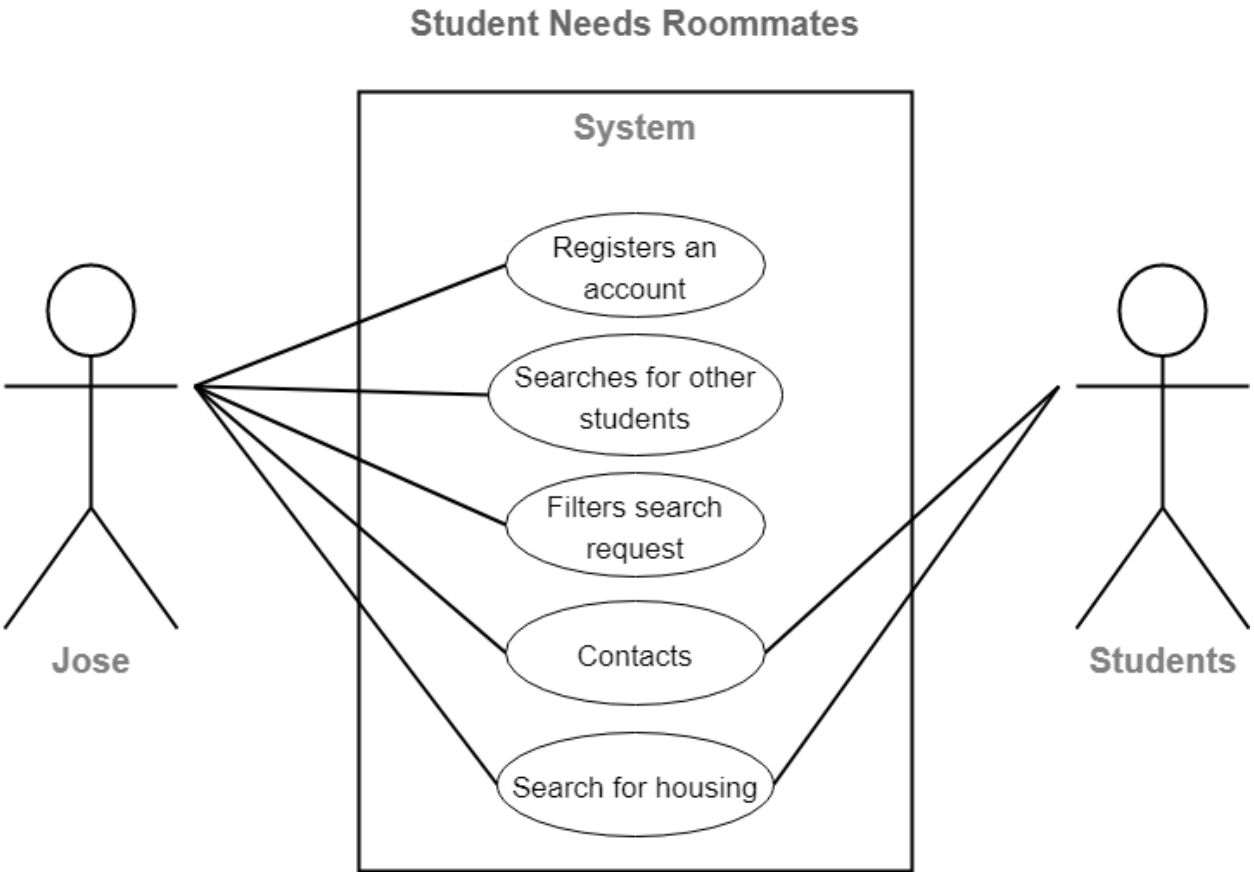
Title:	Creating a Review
Actor(s):	John and Kevin
Description:	John, a student who is renting a unit that is owned by Kevin, a landlord, is having some issues with the in-unit washing machine. In order for John to have the washing machine in his units fixed he had to reach out to Kevin multiple times and felt uncomfortable asking him to resolve the issues multiple times. On completion of his lease agreement, John leaves Kevin a negative landlord review.



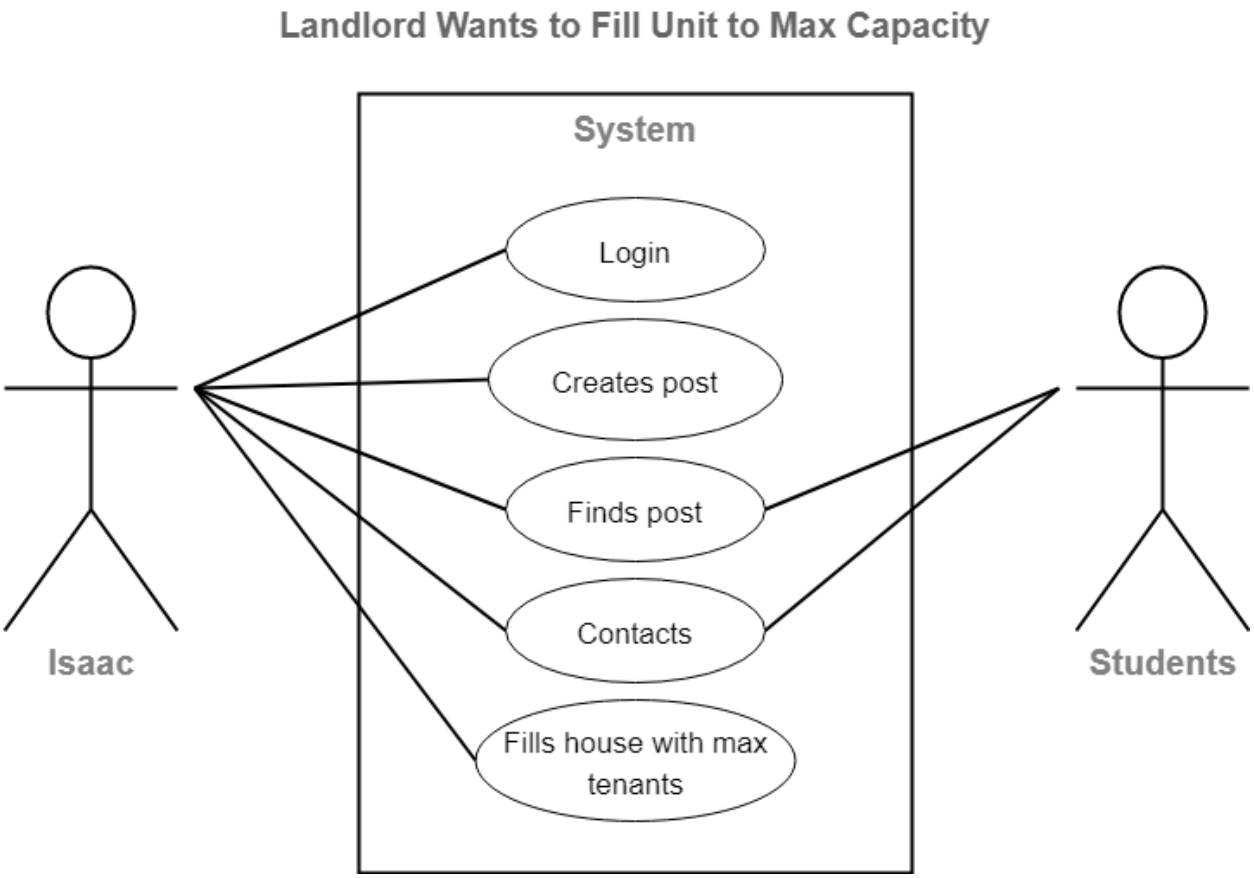
Title:	Student Creates a New Account
Actor(s):	Max
Description:	Max, a computer science student, is unhappy with his current roommate. Max's roommate doesn't have the same interests as him and doesn't even go to university. Max creates an account on DormMates and is able to instantly find other students who are in the same situation as him. In order for him to create a student account he needs to provide his .edu email and verify that he is a student. After that the admin verified his account and proceeded with his search.



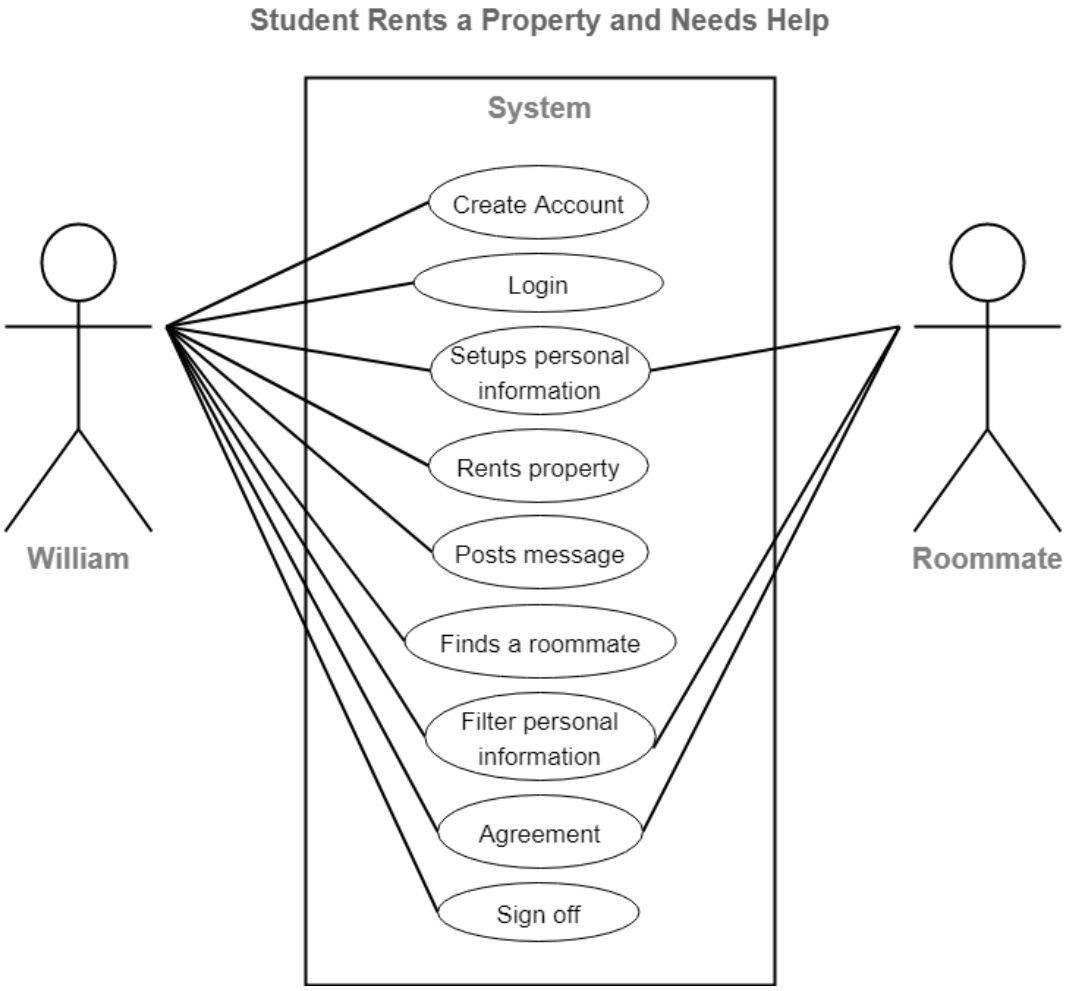
Title:	Student Needs Roommates
Actor(s):	Jose
Description:	An incoming freshmen student, Jose, is searching for a housing option that is within his budget. He is looking for 3 other students to increase the number of available housing options. Jose creates an account on DormMates and searches for roommates that match what he is looking for. He is looking for students that are on a similar schedule as him and have the same major as Jose. He finds and contacts other students that he wants to be roommates with. Jose and the other students search for a housing option within their budget together.



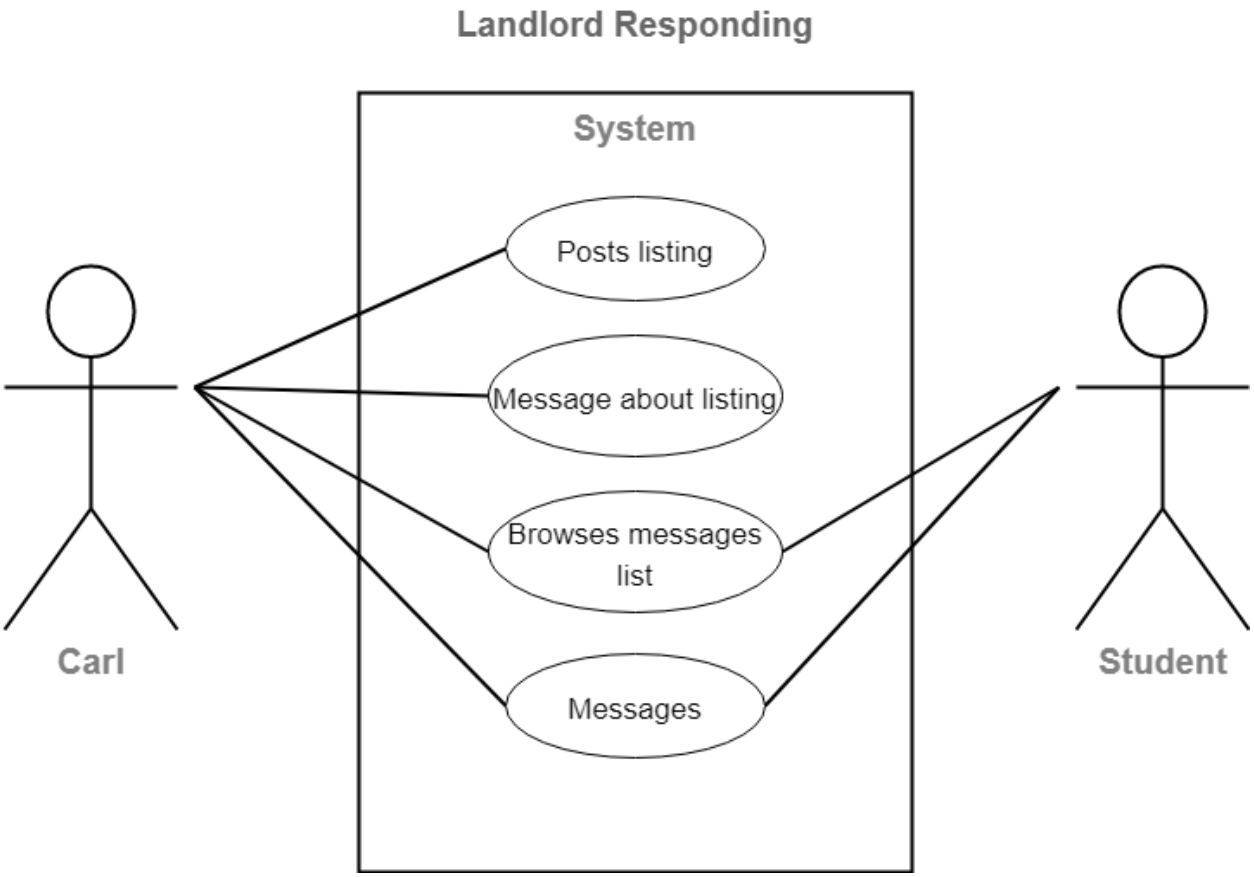
Title:	Landlord Wants to Fill Unit to Max Capacity
Actor(s):	Isaac
Description:	A landlord, Isaac, has logged on to the service and created a listing for his unit with all the available amenities that it offers. It has 3 rooms and can house up to 5 people if 2 students are willing to share a large room. Isaac wants to fill the unit with the maximum number of students that he can. Isaac is contacted by a group of students that found his post and is able to fill out the house with the maximum number of tenants.



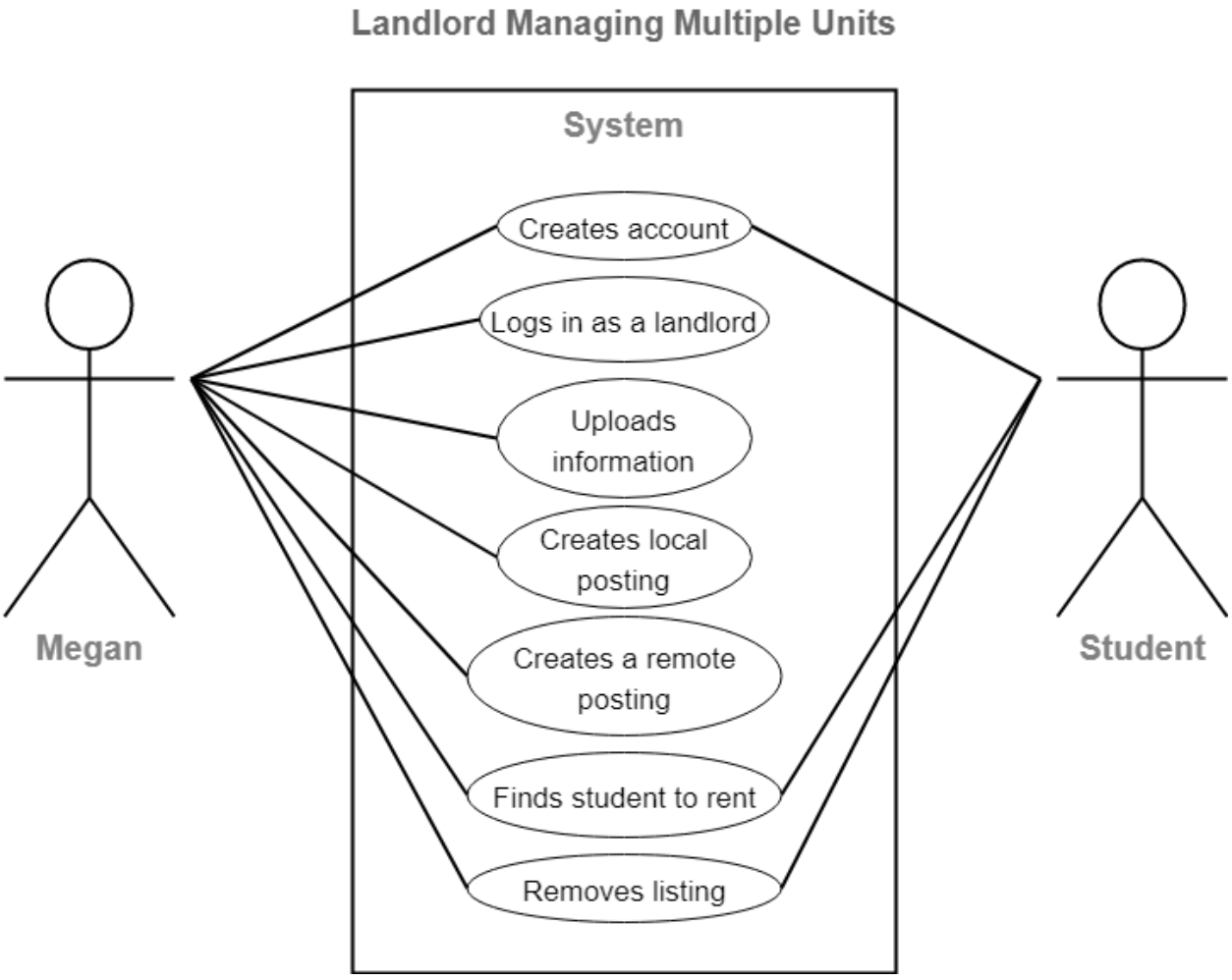
Title:	Student Rents a Property and Needs Help
Actor(s):	William
Description:	The transferring sophomore student, William, went on DormMates to search for housing and a roommate. He needs to find a roommate with a computer science major and attends San Francisco State University. He wants a roommate who can help with his assignments and has a hobby of playing computer games. William then finds a roommate who satisfies his preferences.



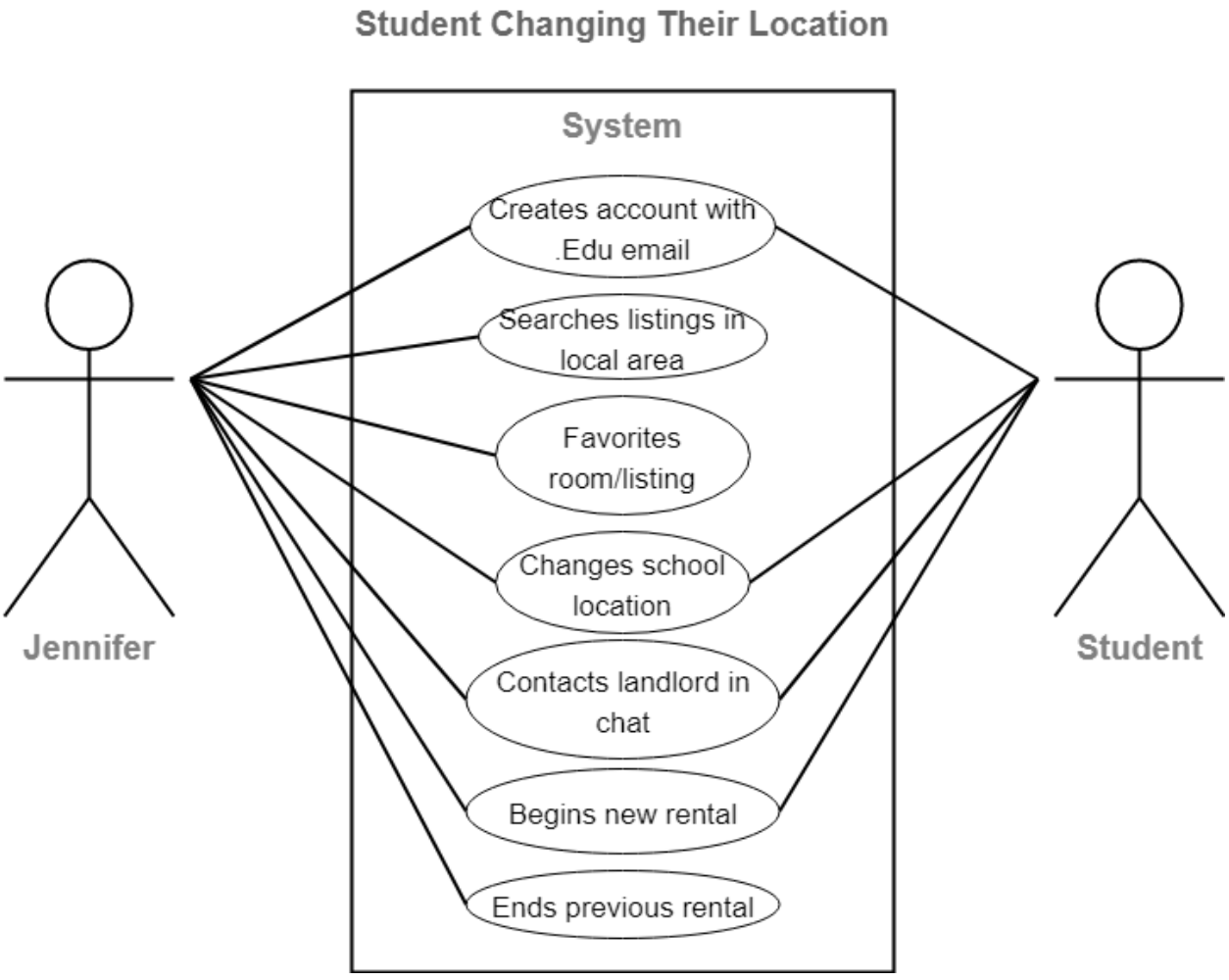
Title:	Landlord Responding
Actor(s):	Carl
Description:	Carl, a landlord who recently listed a rental unit on our application, browses through an array of interested students who messaged him about the listing. Carl is looking for 2 students to rent out his 2 bedroom in-law that are reliable and clean. He takes his time in choosing who to message since he wants students who fit his criteria. Carl decides which students and messages them.



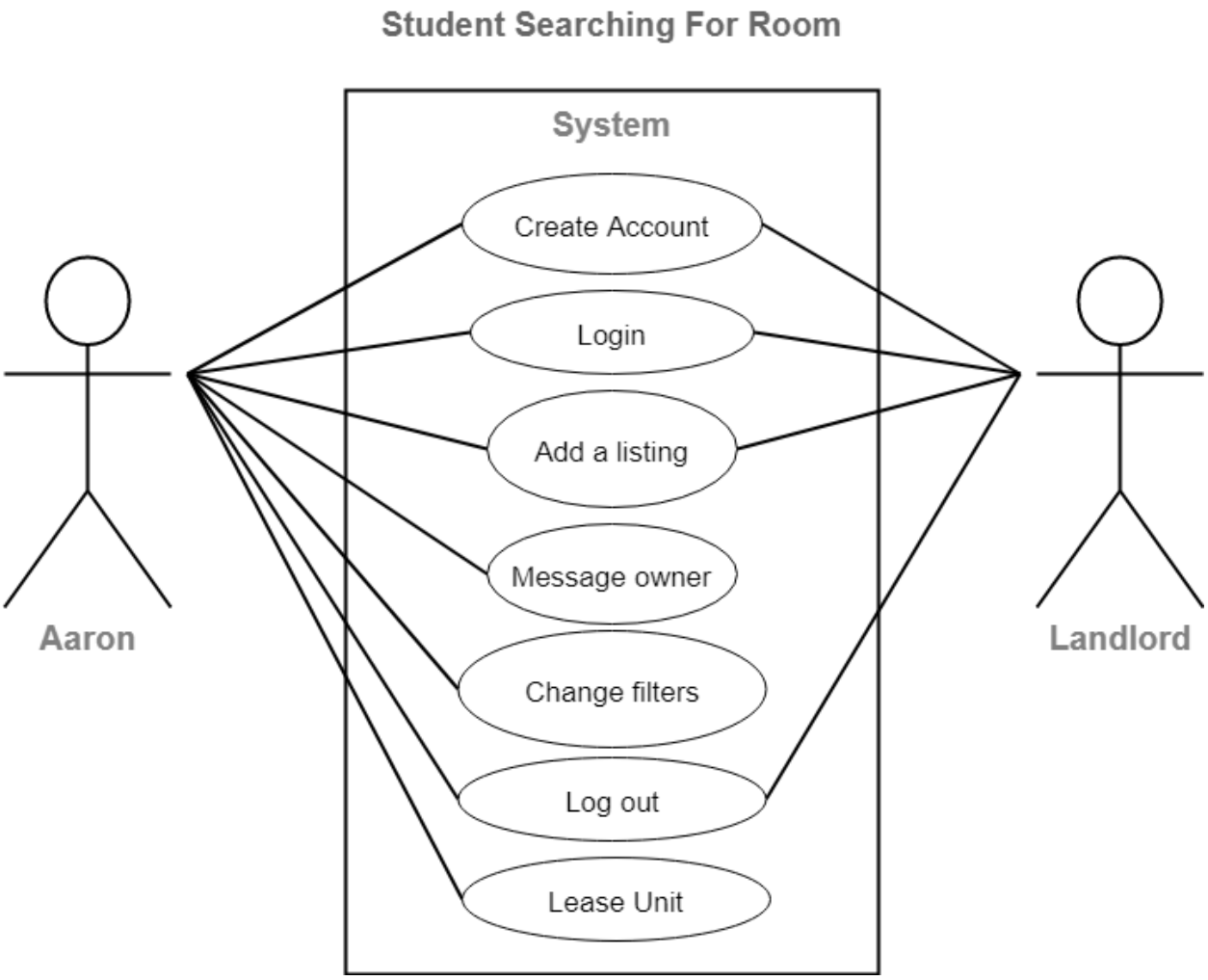
Title:	Landlord Managing Multiple Units
Actor(s):	Megan
Description:	Megan owns a multi unit dorm style building near the downtown area of her city, with two major colleges down the road from her units and more units in another large city in the same state. She creates multiple listings in each city and is able to house students she's housed in different areas and keep connections.



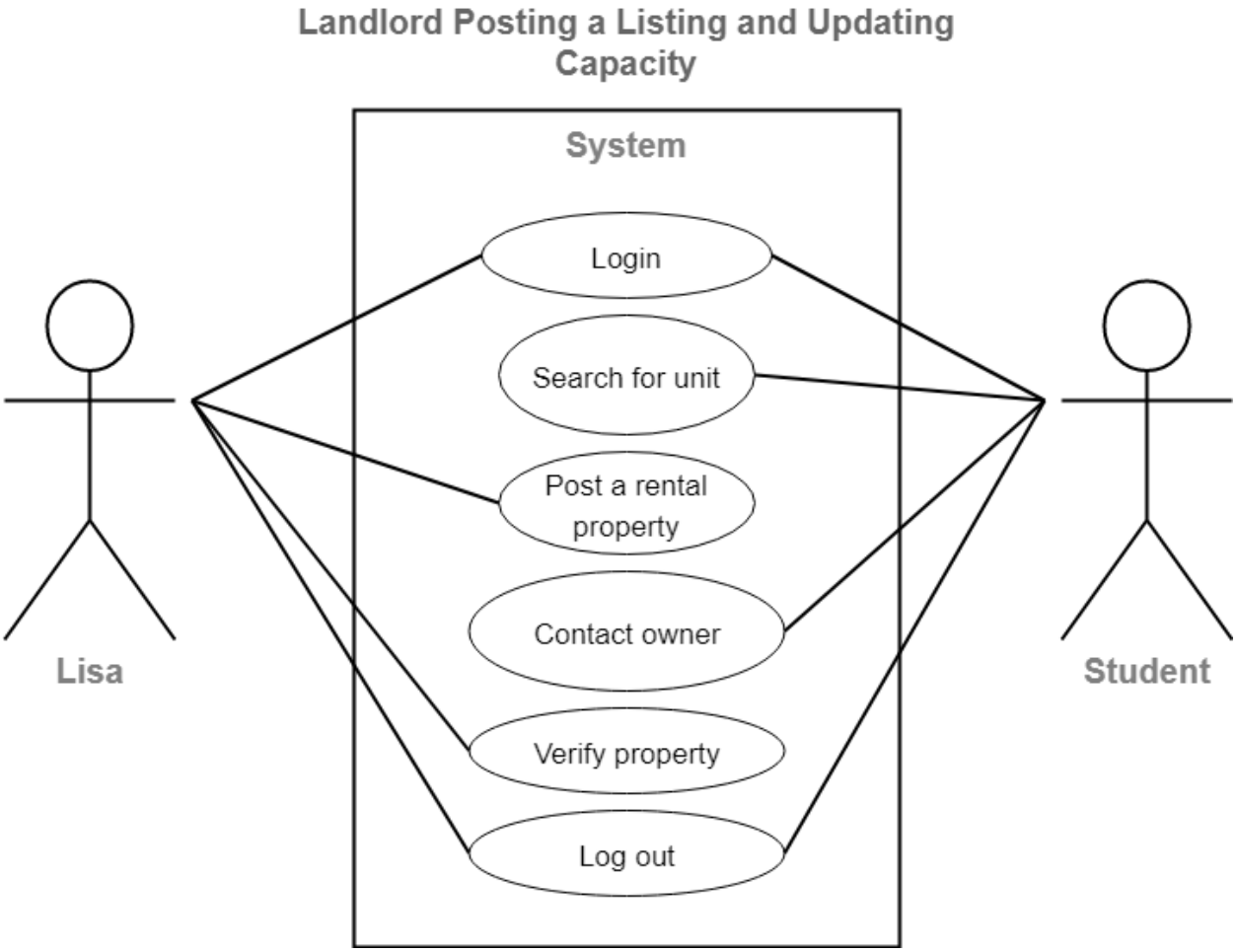
Title:	Student Changing Their Location
Actor(s):	Jennifer
Description:	Jennifer is moving to LA to start school, she is transferring from FAU. She signs up through the portal with her UCLA edu email and is immediately able to see listings in her area according to her student email.She then favorites the ones she likes. This connects her to more roommates that share her interest and degree plan. She was able to meet other UCLA students and live with them.



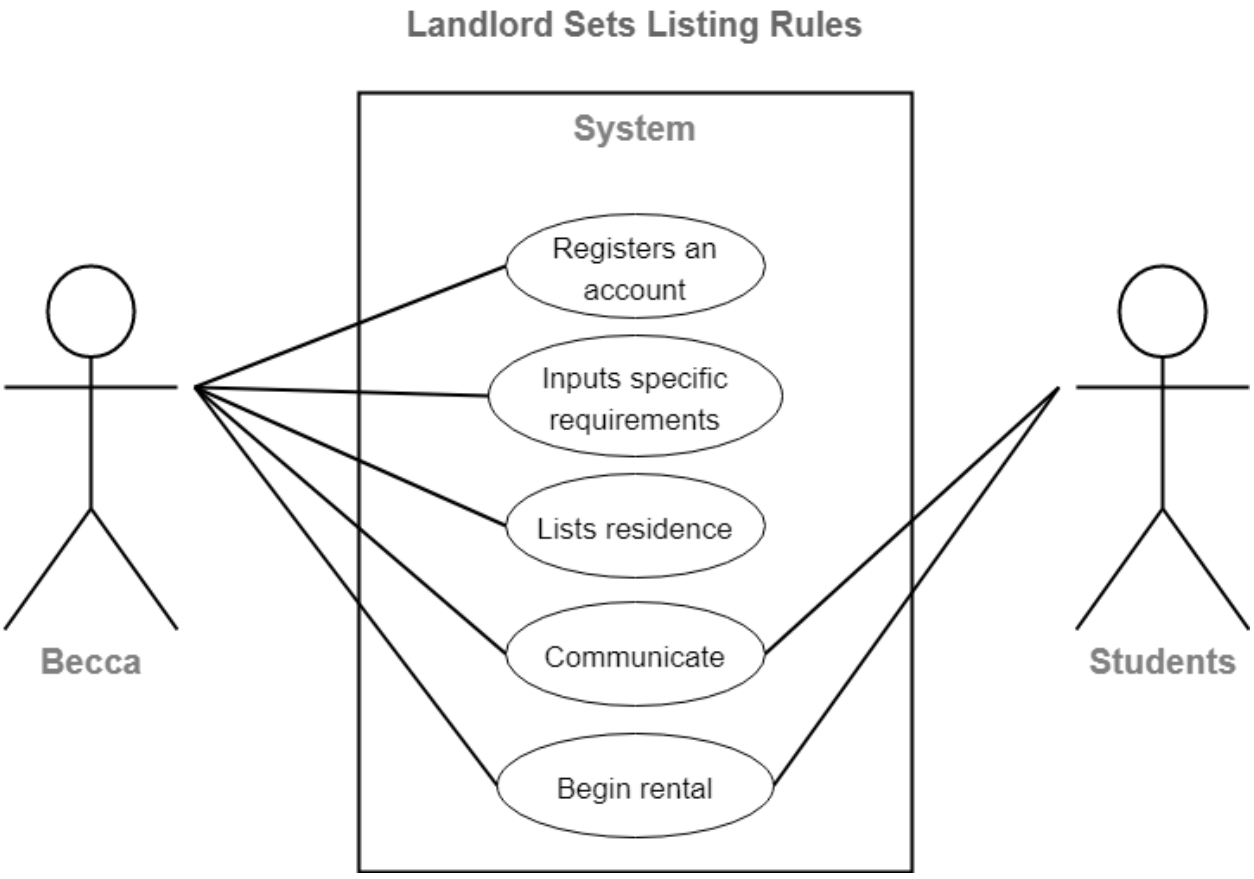
Title:	Student Searching For Room
Actor(s):	Aaron
Description:	Aaron recently moved to San Diego to start school and he is looking to rent a room with one more student near the city. He signs up through the portal with his UCSD account and he can see some listing near him and through his filters, he can look up available rooms in his price range and connect with more students from UCSD.



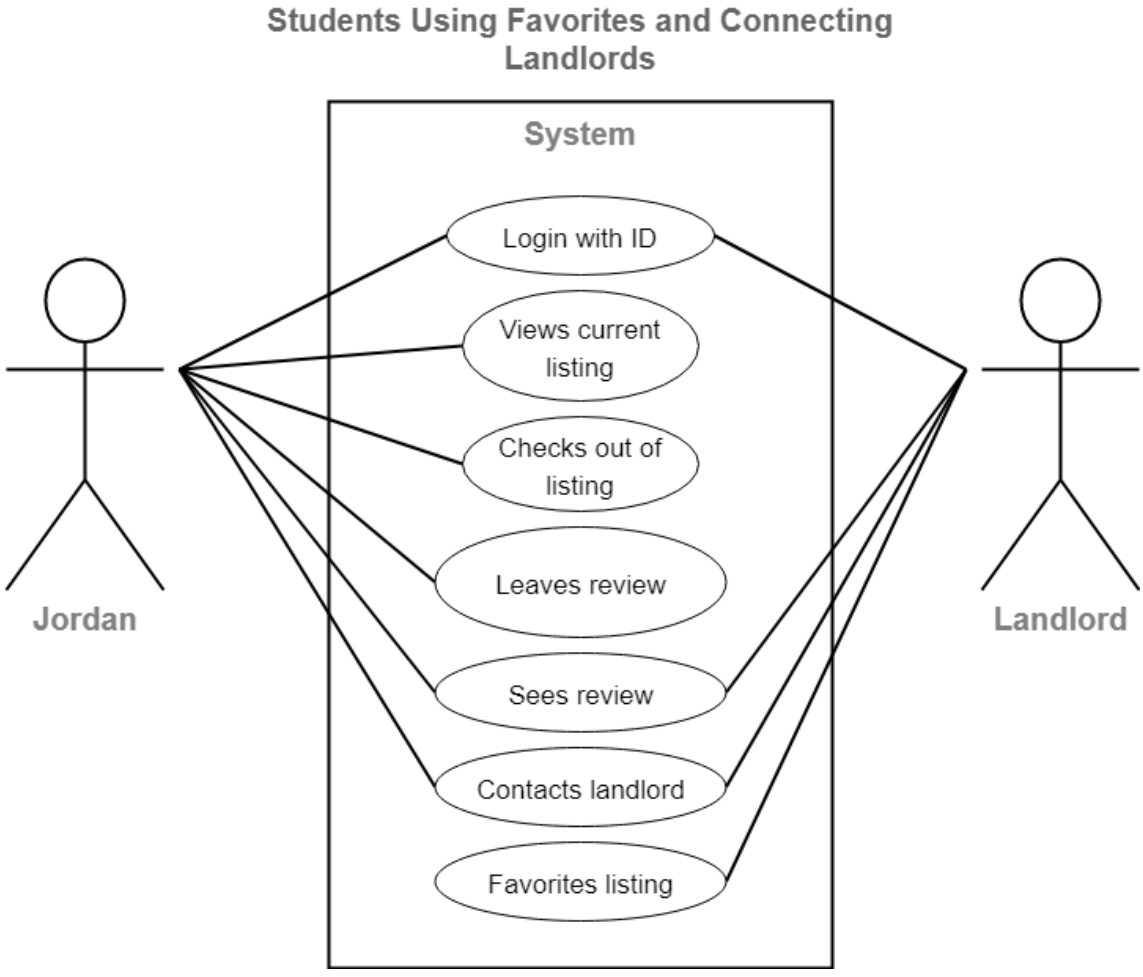
Title:	Landlord Posting a Listing and Updating Capacity
Actor(s):	Lisa
Description:	Lisa owns a small two floor building near the city and she has one room available for rent for two students only. She creates an account through the portal and lists her room and price for each student. After one student messages her and she agrees that the student can move in she can update the listing and change from 2 students to 1 student needed.



Title:	Landlord Sets Listing Rules
Actor(s):	Becca
Description:	Becca is renting out two bedrooms of her three bedroom house to students. Becca also enjoys a peaceful life with her own schedule and does not like a lot of noise around the house. She has had problems with finding two students who can coexist and agree to her rules. Becca lists her property through the portal, with specific requirements such as no guests, no parties; she then communicates with each student to ensure they agree to her rules. The students also talk to each other to make sure they can coexist peacefully.



Title:	Students Using Favorites and Connecting Landlords
Actor(s):	Jordan
Description:	Jordan is moving out of his current apartment, which was full and not posted. He logs in to update information about his living situation. Jordan then leaves a really good review about the landlord. His friend Greg really liked the area Jordan lived in and saw the review about the landlord. He is able to contact the landlord and get information on the listing and favorites for later.



List of Main Data Items and Entities

- Logged in user
    - Either a student or landlord user that is logged in to the website.
- User Type
    - Users can be either Student, Landlord, or Admin.
- Student
    - Users who signed up with a .edu email are denoted as students.
- Landlord
    - Users who signed up without a .edu email are denoted as landlords.
- Admin
    - User that has the ability to moderate the service.
- Guest
    - Users that have not registered to the website.
- Profile
    - Displays key attributes about a user.
- Listing
    - A representation of a housing unit that was posted by a landlord.
- Favorites
    - Specific listings that a user is interested in and wants to bookmark.
- Institution
    - The higher level education that a student is attending.
- Landlord rating
    - A compiled list of reviews and other attributes that rank a landlord.
- Match
    - Represents two users who may be compatible roommates.
- Chat
    - Students can form chats with other students who they are matched with.
    - Students can form chats with landlords of listings they are interested in.
    - Students can share listing links.
- User activity
    - Users have a history of login and chat.
- Map
    - The map is used to display listings.
- Location
    - Specified region that determines which listings are shown to a user.
    - Represents the physical location of a listing.

- Lease
    - Represents an agreement between one or more students and a landlord which signifies that a lease between the parties exists.
- Price
    - Users have to pay for the rent.
- Amenities
    - Represents the key features of a listing.

Functional Requirements

User

Unregistered User

1. An unregistered user can view listings near an institution of their choice.
2. An unregistered user can not view other student users on the platform.
3. An unregistered user can not view other landlord users on the platform.
4. An unregistered user can create a new account.
5. An unregistered user can create a student account.
6. An unregistered user can create a landlord account.
7. An unregistered user can view the Home page.
8. An unregistered user can view the FAQ page.
9. An unregistered user can view the About page.
10. An unregistered user can view the Terms of Service page.
11. An unregistered user can view the Privacy Policy page.

Registered User

12. A registered user should be able to login to with their email and password.
13. A registered user should be able to login to with their username and password.
14. A registered user should be able to logout of their account.
15. A registered user should be able to submit a forgotten password form.
16. A registered user should be able to submit a forgotten email form.
17. A registered user should be able to change their username.
18. A registered user should be able to change their profile photo.
19. A registered user should be able to change their email address.
20. A registered user should be able to change their password.
21. A registered user should be able to change their university.
22. A registered user should be able to change their location.
23. A registered user should be able to change their password.
24. A registered user should be able to change their email.
25. A registered user should be able to change their name.
26. A registered user should be able to change their age.
27. A registered user should be able to change their gender.
28. A registered user should be able to delete a chat.
29. A registered user should be able to favorite a chat.
30. A registered user should be able to verify their account via email.

Student User

31. A student user should be able to edit their personality.
32. A student user should be able to edit their schedule.
33. A student user should be able to edit their hobbies.
34. A student user should be able to edit their major.
35. A student user should be able to edit their social media links.
36. A student user must have an edu email.
37. A student user must have a completed profile.
38. A student user should be able to rate landlords.
39. A student user should be able to message another user.
40. A student user should be able to add other student users to a chat.
41. A student user should be able to search for listings.
42. A student user should be able to view a general location of a listing on a map.
43. A student user should be able to interact with a map to view more details about a listing.
44. A student user should be able to refresh the listings shown on a map.
45. A student user should be able to sort listings by price.
46. A student user should be able to sort listings by the date they were posted.
47. A student user should be able to sort listings by available rooms.
48. A student user should be able to filter listings by distance from university.
49. A student user should be able to filter listings by price.
50. A student user should be able to filter listings by room numbers.
51. A student user should be able to filter listings by washer and dryer being available or not.
52. A student user should be able to filter listings by wifi being available or not.
53. A student user should be able to filter listings by closet space.
54. A student user should be able to filter listings by bathroom count.
55. A student user should be able to filter listings by whether or not they are furnished.
56. A student user should be able to filter listings by whether or not they have a whiteboard.
57. A student user should be able to filter listings by whether or not they have a living room.
58. A student user should be able to filter listings by whether or not they have a kitchen.
59. A student user should be able to filter listings by whether or not they have parking.
60. A student user should be able to filter listings by whether or not they have a patio.
61. A student user should be able to filter listings by whether or not parking is included.
62. A student user should be able to filter roommate selections by personality.
63. A student user should be able to filter roommate selections by schedule.
64. A student user should be able to filter roommate selections by hobbies.
65. A student user should be able to filter roommate selections by major.
66. A student user should be able to filter roommate selections by gender.
67. A student user should be able to filter roommate selections by age.
68. A student user should be able to filter roommate selection by budget.
69. A student user should be able to view other student user profiles.
70. A student user should be able to favorite a listing.
71. A student user should be able to report a listing.

72. A student user should be able to report another student user.
73. A student user should be able to report a landlord user.
74. A student user should be able to view the student dashboard.
75. A student user shall be able to match with another student

Landlord User

76. A landlord user should be able to post listings.
77. A landlord user should be able to edit a listing.
78. A landlord user should be able to delete a listing.
79. A landlord user will be able to add descriptions to listings.
80. A landlord user will be able to view student profiles.
81. A landlord user will be able to respond to student users.
82. A landlord user will be able to see their own reviews.
83. A landlord user will be able to repost their listing once expired.
84. A landlord user will be able to change a listing's price.
85. A landlord user will be able to change a listing's photos.
86. A landlord user will be able to change a listing's location.
87. A landlord user will be able to change a listing's available rooms.
88. A landlord user will be able to change a listing's description.
89. A landlord user will be able to view historical listing data.
90. A landlord user should be able to view the landlord dashboard.
91. A landlord user should be able to pay to promote a listing.
92. A landlord user should be able to pay to post a listing.

Administrator User

93. Admin users should be able to approve listings created by landlords.
94. Admin users should be able to deny listings created by landlords.
95. Admin users should be able to disable landlord user accounts.
96. Admin users should be able to disable student user accounts.
97. Admin users should be able to view user reports.
98. Admin users should be able to view listing reports.
99. Admin users should be able to respond to user reports.
100. Admin users should be able to respond to listing reports.
101. Admin users should be able to edit user profiles.
102. Admin users should be able to feature a listing.
103. Admin users should be able to alter a landlord's rating.
104. Admin users should be able to alter a student's rating.

Non-Functional Requirements

Functionality

1. The website should utilize all tools and frameworks approved by the CTO.
2. The website should be easy to use and intuitive.
3. The website should have a simple and non-cluttered interface.
4. The website's interface should be uniform across all pages.
5. The website should be responsive across all modern devices.
6. The website will use Amazon Web Services for deployment.
7. The website will use Amazon Web Services for its database.
8. The website should use HTTPS for all requests.

Security

9. Users must authenticate themselves before accessing any protected pages.
10. Users must authenticate themselves if their cookie is expired.
11. The student dashboard page must only be available to verified student users.
12. The landlord dashboard page must only be available to landlord users.
13. Registered users should be able to view their own chat messages.
14. Registered users should be able to send messages only within their group.
15. All sensitive information must be encrypted before stored in the database.

Privacy

16. Only registered users will be able to view all listings.
17. Only registered users will be able to view students.
18. Landlords will not have access to viewing other landlords.
19. Landlords will not be able to search student data.
20. Landlords will not be able to search landlords.
21. Registered users' chat messages should remain private.

Legal

22. All users must accept the terms and service policy before creating an account.
23. All users must accept the privacy policy before creating an account.
24. All landlords must prove ownership of a listing.
25. The website must have a copyright notice.
26. The website must have a privacy policy notice.
27. The website must have a terms and conditions notice.
28. The website must have a cookie notice.
29. All content uploaded to the site must be owned by the user who is uploading it.

Performance

30. The frontend must have processes in place that prevent it from being offline.
31. The backend must have processes in place that prevent it from being offline.
32. The website load time should be within industry standard requirements.

System Requirements

33. The website shall work up to Version 91.0.4472.106 Google Chrome.
34. The website shall work up to Version 14.0.03 Safari.
35. The website shall work up to Version 91.0.864.48 Microsoft Edge.
36. The website shall work up to Version 85.0 of Mozilla Firefox.
37. The website shall work up to Version 11.0 of Android.
38. The websites shall work up to Version 14.6 of IOS.
39. The website will be supported in English language.

Marketing

40. The website should follow SEO best practices.
41. Each page on the website shall have the logo next to the navigation bar.
42. Each page will be clear and easy to navigate for new visitors.
43. Each user shall be able to connect their account with their social media platforms.

Content

44. The website will have a navigation bar.
45. The website navigation bar will direct users to different pages on the website.
46. The website pages will have a footer.
47. The website will have a scalable map.
48. The website should give registered users the option to private message.

Scalability

49. The website should be capable of handling a large number of listings.
50. The website should be composed of a frontend and backend which are separate codebases.
51. The website shall be able to handle a large number of users.
52. The chat rooms shall be able to handle a large number of users.

Capability

53. The website should process all requests as expected by the user.
54. The website should respond with a descriptive error if one occurs.
55. The website should alert users when they are about to leave the site.

Look and Feel

56. The navigation bar should have a logo.
57. The navigation bar should have a plain dark color background.
58. The navigation bar should have a light shade hover color button.
59. The footer should have a logo.
60. The footer should have a sitemap with all site pages.
61. The website should have a plain color layout.
62. The website should have a simple layout.
63. The website will have a readable font.
64. The website elements fonts will be uniform.
65. The website elements will be continuous.
66. The website's pages will be scrollable in the vertical axis.
67. The font should be roman new times.
68. The feeling should be friendly.
69. The website should not be repetitive.
70. The website should be easy to traverse.
71. Pages should be instant loaded.
72. The private account page should be easy to find.
73. The private chat should be easily identifiable.
74. The private chat font will be easy to read and uniform.
75. The map should be easily identifiable.
76. The map key will be easily identifiable.
77. Profiles will clearly display a user’s role.
78. The post should be easily identifiable.
79. The forum should be easily identifiable.
80. The buttons should be easily identifiable.
81. Listing filters should be easily identifiable.

Coding Standards

82. All code must be reviewed before it is merged with any of the three main branches.
83. All code must be submitted via pull requests.
84. All code must be pushed to proper branches.
85. All code should be documented.
86. All code should be organized.
87. There should be no repetitive code.
88. There should be no unused code.
89. All code should have in-line comments when needed.
90. The code should have a uniform formatting style.
91. The backend code should be an object-oriented programming paradigm.
92. The backend must implement methods to prevent SQL injection.

Availability

93. The frontend must be online at all times.
94. The backend must be online at all times.
95. The website is updated if and only if code is pushed to the master branch.
96. The website will resync if a loss of connection occurs.
97. The website shall display error messages when errors occur.
98. The website will be managed on a PST time zone.

Cost

99. Amazon web service's server is free.
100. Amazon web service relational database is free.
101. Server must not exceed the free tier.
102. Server maintenance is free.

Storage

103. Store users profile data on the database.
104. Store landlords listings on the database.
105. Remove listings from the database after it has been deleted by the user.
106. Store up to 60 days of inactive listings(incase user wants to repost)
107. Remove listings from the database after 60 days of inactivity.
108. Repost will restart the 60 day clock of storage time.
109. Store students' chat history on the database.
110. Store usernames on the database.
111. Store emails on the database.
112. Store passwords on the database.
113. Store landlord information on the database.
114. Store landlord photos on the database.
115. Store Students photos on the database.
116. Store error logs on the database.

Expected Load

117. The website will be able to handle as many users as Amazon Web Services can support.
118. The website will be able to handle as many listings as Amazon Web services can support.

Competitive Analysis

	Craigslist URL	**Roomiematch** URL	**Roommates** URL	**Facebook** URL	**Forrentuniversity** URL
Strengths	• Intuitive • Simple • Many listings • Many filters • Speed • Save listings • Cross platform	• Security and safety focus • Human moderation	• Identity verification • Profile matching • Easy to use • Cross platform	• Popular • Organized • Easy to use • Data integrated Groups • Dedicated community	• Easy to use • Strong search system • Lists apartments and houses
Weaknesses	• Unregulated • No internal chat • Repetitive listings • Lacks design • Not visual based	• Locked behind paywall • Only focused on roommates • Website not intuitive	• Unclear listings • Listings locked by paywall	• Forums posts from top to bottom • No specific search for forums on page	• Generic landing page • Only focused on listings • No chat or student features • Not moderated • Unsatisfying user experience
Pricing	• $5 apartment listings in Boston, Chicago, and NYC areas	• $19.95 per year for pro roommate search	• $6/3 day trial • $20/month • $30/2months	• Free	• Free to post listings
Target Market	• Anyone	• Anyone searching for roommates	• Anyone searching for roommates	• Anyone	• University students
Onboarding experience	• Simple and fast	• Complicated • Too many questions	• Lots of steps • Too many questions	• Simple and fast	• None

Feature	**Craigslist** URL	**Roomiematch** URL	**Roommates** URL	**Facebook** URL	**Forrentuniversity** URL	**DormMates**
Shows user activity	+	-	++	++	-	++
Requires .edu email	-	-	-	-	-	++
Chat	-	+	+	++	-	++
Landlord rating system	-	-	-	-	-	++
Roommate matching	-	+	+	-	-	++
Moderated listings	-	++	+	++	-	++
Map	+	-	+	++	++	++
Listing filter	+	-	-	++	+	++

Legend: - Feature does not exist, + Feature exists, ++ Feature is superior

Summary of Competitive Analysis

When researching current companies in this market, we were not able to find any service that caters directly to university students. While some services had some of the features DormMates seeks to offer, none had all of them in one easy-to-use platform. According to our research, we found three common themes across all of our competitors: usability, price, and student-focused features. While some competitors do prioritize usability, it does not seem that it is a main focus for our competitors and most did not have a simplistic and intuitive interface. In addition to this, competitors also placed a large emphasis on turning a profit. All but one of our competitors offered some type of paid service or locked key features such as verified listings and roommate matching behind a paywall. Lastly, none of the competitors on the market cater directly to students through student-focused features and we believe that this opens up a huge opportunity for us.

DormMates believes that by focusing on the three themes of student-focused features, usability, and price we can create a service that has a substantial edge over our competitors. Students will be at the core of our service and everything they see will be based on the university they are attending. All roommates that student users see on our platform will go to the same university as them and have similar interests. Similarly, all listings that students see will be near their university. In terms of usability, we will place the user-experience above all else by creating a simple and intuitive website that allows our users to find exactly what they are looking for. When it comes to price, we will not lock down any “pro” features behind a paywall. We believe that by allowing students to use all of our features we can generate a profit by retaining the most amount of users possible. The thing that allows us to generate a profit are the landlords. We will allow landlord users to pay us a fee in order to bring more attention to their listings through the user of featured listings. In addition to this, as our student user-base grows, we plan on taking a fee for all listings that are posted on our platform.

High Level System Architecture and Technologies

Hosting

- Server Host: AWS EC2 1vCPU 1GB RAM
- Database Host: AWS RDS T2 Micro (MySQL 8.0.20)
- DNS: Cloudflare
- Email API: Mailjet

Server Software Stack

- Operating System: Ubuntu Server 20.04 LTS
- Web Server: NGINX 1.20.1
- Server-Side Language: Javascript
- Server Framework: Node.js 10.19.0
- Server Process Manager: PM2 5.1.0

Frontend Technologies

- Web Framework: Express 4.17.1
- CSS Framework: Bootstrap 5.0.0
- Templating Engine: Pug 3.0.2
- Interactive Maps Library: Leaflet 1.7.1
- Icons Library: FontAwesome 5.15.3

Backend Technologies

- Backend Framework: Express 4.17.1
- SSL Cert: Cloudflare
- Encryption: Bcrypt 5.0.1
- Sessions: express-session 1.17.2
- Sessions MySQL Store: express-mysql-session 2.1.6
- Image Middleware: Multer 1.4.2
- Real-time Communication: pusher-js 7.0.3
- Academic Email Middleware: Academic-Email-Verifier 3.0.3
- Email sending library: node-mailjet 3.3.4
- Authentication: Passport 0.4.1 (Local Strategy)

Additional Technologies

- IDE: vsCode, Sublime Text
- Web Analytics: Google Analytics

Checklist

Item	Status
Team found a time slot to meet outside of class	DONE
GitHub master chosen	DONE
Team decided and agreed together on using the listed SW tools and deployment server	DONE
Team ready and able to use the chosen back and frontend frameworks and those who need to learn are working on learning and practicing	DONE
Team lead ensured that all team members read the final M1 and agree/understand it before submission	DONE
GitHub organized as discussed in class (e.g. master branch, development branch, folder for milestone documents, etc...)	DONE

List of Team Contributions

Andrei

- Scheduled and conducted 5x meetings where the team worked on M1together.
- Worked with M1 editor by giving advice on how to format certain areas of the document.
- Worked with the team to create the executive summary.
- Worked with the team to create main use cases.
- Worked with the team to list maindata items and entities.
- Worked with the team to create initial list of functional requirements.
- Worked with the team to create list of non-functional requirements.
- Worked with the team to create the competitive analysis.
- Worked with backend lead to list the high-level system architecture and technologies.
- Created the checklist and made sure everything on the checklist was taken care of.
- Compiled the list of team contributions and complaints.
- Made sure that the entire team looked over the final M1 document and agree/understand it.

Meeka

- Attended all meetings that were scheduled by the team lead.
- Participated in each task discussed and written during the meetings such as the executive summary, main use cases, main data items & entities, functional requirements, non-functional requirements, and competitive analysis with the team together.
- Revised and edited the draft documents with assistance from the team lead.
- Created necessary diagrams and tables for the final document.
- Created and formatted the final document as the M1 editor.

Jonathan

- Task 1: Executive Summary:

The group together contributed to this, we all wrote it together and edited the paragraph. I contributed roughly 1/5 of this part along with the team. I edited some sentences and contributed some. We all collectively did this.

- Task2: Main Use Cases:

The team was assigned multiple use cases, I contributed 3 use cases and diagrams for the cases. I added a use case regarding 3 features. Landlord listings, Student use cases and Favorite feature. Roughly a paragraph each.

- Task 3: List of main data items and entities:

This task we worked as a team to finish. I contributed about 6 entities with the team as we brainstormed. Users/ Students/Landlords/Locations. The team collectively edited them too.

- Task 4: Initial list of Functional Requirements:

This task was a group effort during our meeting. We were all tasked with making a list of items and then compiled them together. I contributed roughly 30 functional requirements. These requirements were also edited later in the document.

- Task 5: List of Non functional requirements

This task was also a group effort during meetings. I contributed another 20~30 non functional requirements for this document. We were all tasked with coming up with a list. I made about 40 items but only 20 made it onto the list.

- Task 6: Competitive Analysis:

This task was assigned to the team to compile and come up with an analysis. This task was mostly done by the team. I contributed verbally as they edited the document. I did not contribute/ write on the document as I was assigned Task 7. We all did talk about each item and column.

- Task 7: High level Architecture

I was tasked with this, I met up with the Team Leader and we discussed these technologies and researched on how to implement them. These were decided before hand and Andrei and I talked more in detail about what items we wanted to use and organized them. I did discuss some features we will need to consider, and later on Andrei found some more. So I'd say this task was done 50/50 by Andrei and myself.

Alexandre

- Helped brainstorm ideas to be used for Task 1 Executive Summary
- Helped complete Task 1 Executive Summary
- Created Main Use Cases to help complete Task 2 Main Use Cases
- Helped determine the Data Items and Entries for Task 3 Data Items and Entries completion
- Helped figuring out what would the Functional and Non-Functional requirements are to complete Task 4 Functional Requirements and Task 5 Non-Functional Requirements
- Researched competitors to compare our product for Task 6 Competitive Analysis
- Helped create explanation of wh

Jimmy

- I met the team leader's agenda everyday by responding with emoiji on discord.
- I usually prepare ideas before the tasks so that I can talk to my group.
- I do respond to my group when they ask who did what before.
- For task 1, I wrote a paragraph and bullet points for brainstorm's executive summary.
- I had never missed a meeting and respond to my team leader's who contribute what.
- I wrote the entire non-functional's idea and some ideas for functional's idea for brainstorm.
- For task 2, I wrote a use case in a paragraph and a case diagram.
- I went to checked with team leader if my case diagram is acceptable and appropriate.
- For task 3, I went to wrote user activity and map for main data items and entities.
- I came up idea about agreement for lease.
- For task 4, I contribute one fourth of the functional requirement.
- For task 5, I contribute one third of the non-functional requirement.
- I went to question backend code format and front end looks.
- For task 6, I wrote the almost entire important points for facebook for competitive analysis.
- I went to talk about what the competitor does.

Sayed

- Contributed on non-functional requirements
- Contributed on competitive analysis doing research forrentuniversity.com
- Contributed on functional requirement
- Helped on topic brainstorm executive

Ahad

- Worked on rewriting and rewording the executive summary
- Worked with the team and contributed multiple functional requirements
- Worked with the team and contributed multiple non-functional requirements
- Worked on researching and completing the competitive analysis for all five competitors
- Researched and provided data for competitor features
- Tested competitors website processes and account creating flow

SW Engineering CSC648 Summer 2021

DormMates

Milestone 2 • Version 2 • 08 July 2021

Team 01

Team Lead and Github master	Andrei Georgescu	ageorgescu@mail.sfsu.edu
Frontend Lead	Meeka Cayabyab	
Backend & Database Lead	Jonathan McGrath	
Engineer	Alexandre Ruffo	
Engineer	Jimmy Yeung	
Engineer	Sayed Hamid	
Engineer	Ahad Zafar	

History Table

Version	Date	Notes
M2V2	7/13/2021	Updated database business rules, functional requirements, uml diagram, application and deployment diagram.
M2V1	7/08/2021	Initial submission
M1V2	07/01/2021	Added database master role to the title page, addressed use cases feedback, addressed functional requirements feedback, and addressed competitive analysis feedback.
M1V1	6/22/2021	Initial submission

Table of Contents

Table of Contents 3

Data Definitions 4

Prioritized Functional Requirements 7
- Priority 1 - Required 7
- Priority 2 - Desired 8
- Priority 3 - Opportunistic 9

UI Mockups and Storyboards 10
- UI Mockups 10
- Storyboards 10

High Level Database Architecture and Organization 24
- Business Rules 24
- Entities 25
- ERD Diagram 28
- Database Model 29
- DBMS / Information 30

High Level APIs and Main Algorithms 31

High Level UML Diagrams 35

High Level Application Network and Deployment Diagrams 37
- Network Diagram 37
- Deployment Diagram 38

Key Risks 39
- Skills 39
- Schedule 39
- Technical 39
- Teamwork 40
- Legal/Content 40

Project Management 41

Detailed List of Contributions 42

Data Definitions

- **General User**: A user who can view and access the website limited to the website's features. A general user must **register** by creating a student or landlord account to gain more privileges.
    - **Registration**: A general user has the option to create an account.
    - A general user shall be able to view limited listings based on an institution.
    - A general user shall be able to view the informational pages regarding the service.

- **Registered User**: A user who has access to the website's core features.
    - **Account Contains:** All registered user accounts must contain the following information.
        - **Username**: A username is required to create an account.
        - **Email**: A unique email is required to create an account.
        - **Password**: A password that will be hashed for security purposes is required.
        - **Accepted Terms of Use**: Must accept to create an account.
        - **Name:** A registered user must have a first and last name.
        - **Gender:** A registered user must specify a gender.
        - **Age:** A registered user must have an age.
        - **Photo**: A profile shall have a picture of the registered user.
    - Has the ability to log into the service.
        - **Email**: Needs to login with a unique email.
        - **Username**: Needs to login with a unique username.
        - **Password**: Needs to provide a password.
    - Has the ability to communicate with other users.
    - Has the ability to create a student account.
    - Has the ability to create a landlord account
    - **Student:** A user who signed up with a .edu email is denoted a student.
        - Has the ability to create a group.
        - Has the ability to invite other student users to a group.
        - Has the ability to invite landlords to groups.
        - **Profile:** A profile will display information / attributes of a student user.
            - **Institution :** An institution will be part of a profile
            - **Major:** A major will be part of a student profile
            - **Schedule:** Needs to specify if they have a night or day schedule
            - **Hobby**: A hobby will be part of a student profile.
            - **Personalities:** A personality will be part of a student profile.
            - **Social Media:** A link to social media outlets.

    - **Landlord:** A user who signed up with a non .edu email is denoted as a landlord.
        - Has the ability to post new listings.
        - Has the ability to create leases.
        - Has the ability to add students to a lease.
        - **Profile:** A profile will display information / attributes of a landlord.
            - **Listings:** The listings that they have posted
            - **Rating:**A list of ratings given by students.
            - **Student Ratings**: Number of students that have rated the landlord.

- **Admin User**: User that has the ability to moderate the service.
    - Can validate listings and make them public on the platform.
    - Can remove listings from the platform.
    - Can answer reports made by other users on the platform.
    - **Account Contains:** All registered user accounts must contain the following information.
        - **Username**: A username is required for all admin users.
        - **Email**: A unique email is required for all admin users.
        - **Password**: A password that will be hashed is required for all admin users.

- **Listing:** A representation of a housing unit that was posted by a landlord.
    - **Location:** Represents the physical location of a listing.
        - Represented by latitude and longitude coordinates.
    - **Photos:** Represents visual representations of a listing.
    - **Lease:** Represents an agreement between one or more students and a landlord which signifies that a lease between the parties exists.
        - Landlords can create leases for their listings.
        - Students can sign leases.

    - **Amenities:** Represents the key features of a listing that are included.
        - Washer and Dryer
        - Wifi
        - Closets
        - Bathrooms
        - Furnished
        - Whiteboard
        - Living room
        - Kitchen
        - Patio
        - Parking

- **Favorite**: Specific listings that a user is interested in and wants to bookmark.

- **Match:** A representation of two student users who may be compatible roommates.

- **Groups:** A representation of a group of students that are matched and looking for listings together.
    - **Chat:** A way groups can communicate with each other on our platform.
        - Student users can share listing links in chat.
        - Student users can add a landlord user to a chat.
        - Landlord users can create a lease in a chat and invite all student users to sign the lease.

Prioritized Functional Requirements

---

1. Priority 1 - Required

    1.1. Unregistered User

        1.1.1. An unregistered user can create a new student or landlord account.

1.1.2. An unregistered user can view listings near an institution of their choice.
1.1.3. An unregistered user can view the Home page.
1.1.4. An unregistered user can view the Terms of Service page.
1.1.5. An unregistered user can view the FAQ page.
1.1.6. An unregistered user can view the Features pages.
1.1.7. An unregistered user can view the About page.

1.2. Registered User
1.2.1. A registered user should be able to login to with their username and password.
1.2.2. A registered user should be able to logout of their account.
1.2.3. A registered user should be able to submit a forgotten password form.
1.2.4. A registered user should be able to submit a forgotten email form.
1.2.5. A registered user should be able to change their username.
1.2.6. A registered user should be able to change their email address.
1.2.7. A registered user should be able to change their password.

1.3. Students
1.3.1. A student user must have a verified edu email.
1.3.2. A student user must have a completed profile.
1.3.3. A student user shall be able to view the student dashboard.
1.3.4. A student user shall be able to view student user profiles.
1.3.5. A student user shall be able to message another user.
1.3.6. A student user should be able to add other student users to a group.
1.3.7. A student user should be able to add a landlord to a group.
1.3.8. A student user shall be able to search for listings.
1.3.9. A student user shall be able to edit their personality.
1.3.10. A student user should be able to edit their schedule.
1.3.11. A student user should be able to edit their hobbies.
1.3.12. A student user shall be able to filter listings by price.
1.3.13. A student user shall be able to filter listings by amenities.
1.3.14. A student user shall be able to filter listings by distance from university.
1.3.15. A student user shall be able to filter roommate selections by personality.
1.3.16. A student user shall be able to filter roommate selections by major.
1.3.17. A student user shall be able to filter roommate selections by hobbies.
1.3.18. A student user shall be able to filter roommate selections by schedule.
1.3.19. A student user should be able to view the location of a listing on a map.
1.3.20. A student user shall be able to favorite a listing.

1.4. Landlords
1.4.1. A landlord user shall be able to view the landlord dashboard.
1.4.2. A landlord user shall be able to view their own profile.
1.4.3. A landlord user shall be able to post listings.
1.4.4. A landlord user shall be able to edit their listings.
1.4.5. A landlord user shall be able to view student profiles.
1.4.6. A landlord user will be able to respond to student user messages.
1.4.7. A landlord user shall be able to delete their listings.

1.5. Admin
1.5.1. Admin users should be able to approve or deny listings created by landlords.

2. Priority 2 - Desired

2.1. Unregistered User

2.2. Registered User
2.2.1. A registered user should be able to delete a chat.

2.3. Students
2.3.1. A student user should be able to change their university.
2.3.2. A student user shall be able to rate landlords.

2.4. Landlords
2.4.1. A landlord user shall be able view historical listing data.
2.4.2. A landlord user shall be able to repost their listing once expired.

2.5. Admin
2.5.1. Admin users should be able to edit registered user profiles.

3. Priority 3 - Opportunistic

3.1. Unregistered User

3.2. Registered User

3.3. Students
- 3.3.1. A student user shall be able to report a listing.
- 3.3.2. A student user shall be able to report another user.

3.4. Landlords
- 3.4.1. A landlord shall be able to report a listing.
- 3.4.2. A landlord shall be able to report another user.

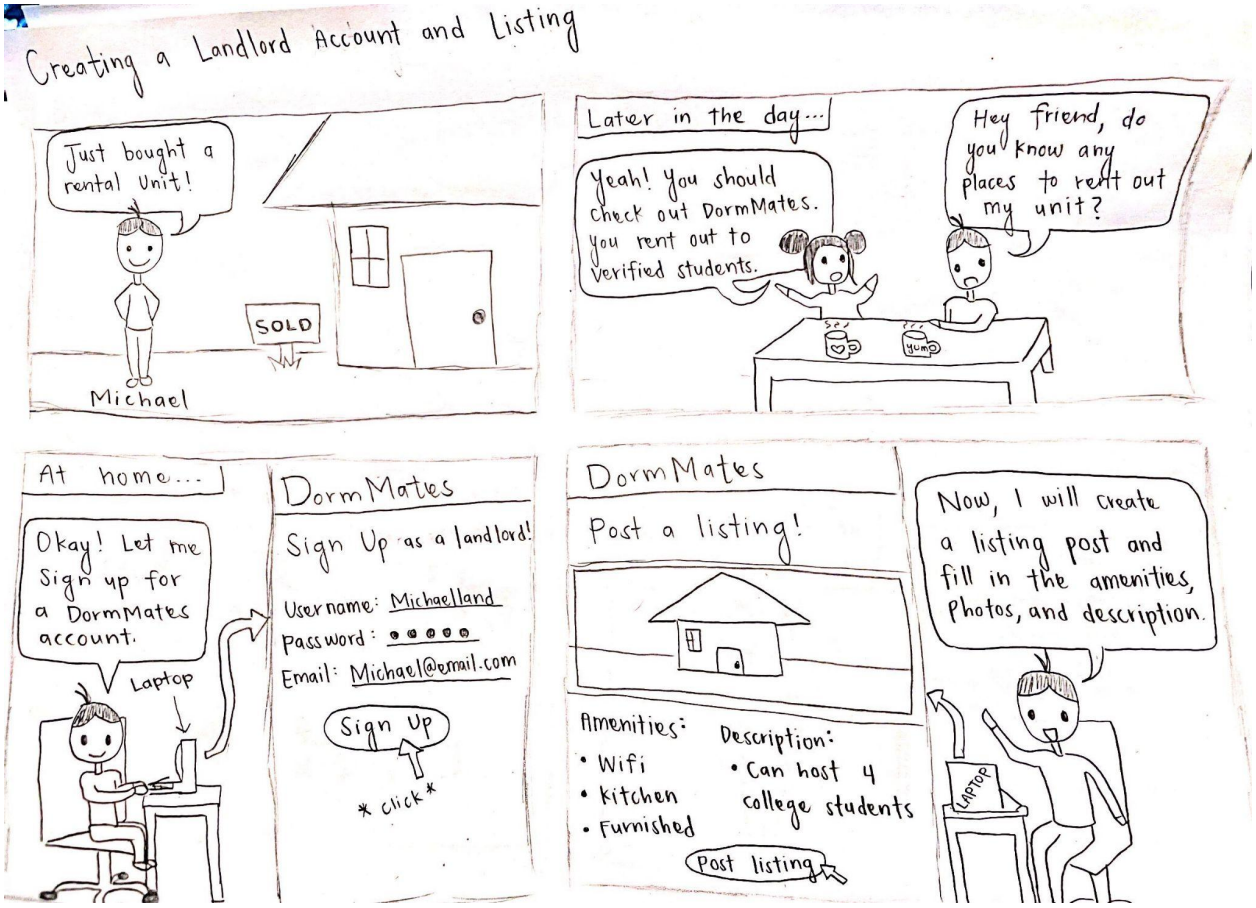
3.5. Admin
- 3.5.1. Admin users should be able to disable non-admin user accounts.
- 3.5.2. Admin users should be able to remove a listing.
- 3.5.3. Admin users should be able to view reports.
- 3.5.4. Admin users should be able to respond to reports.

UI Mockups and Storyboards

UI Mockups

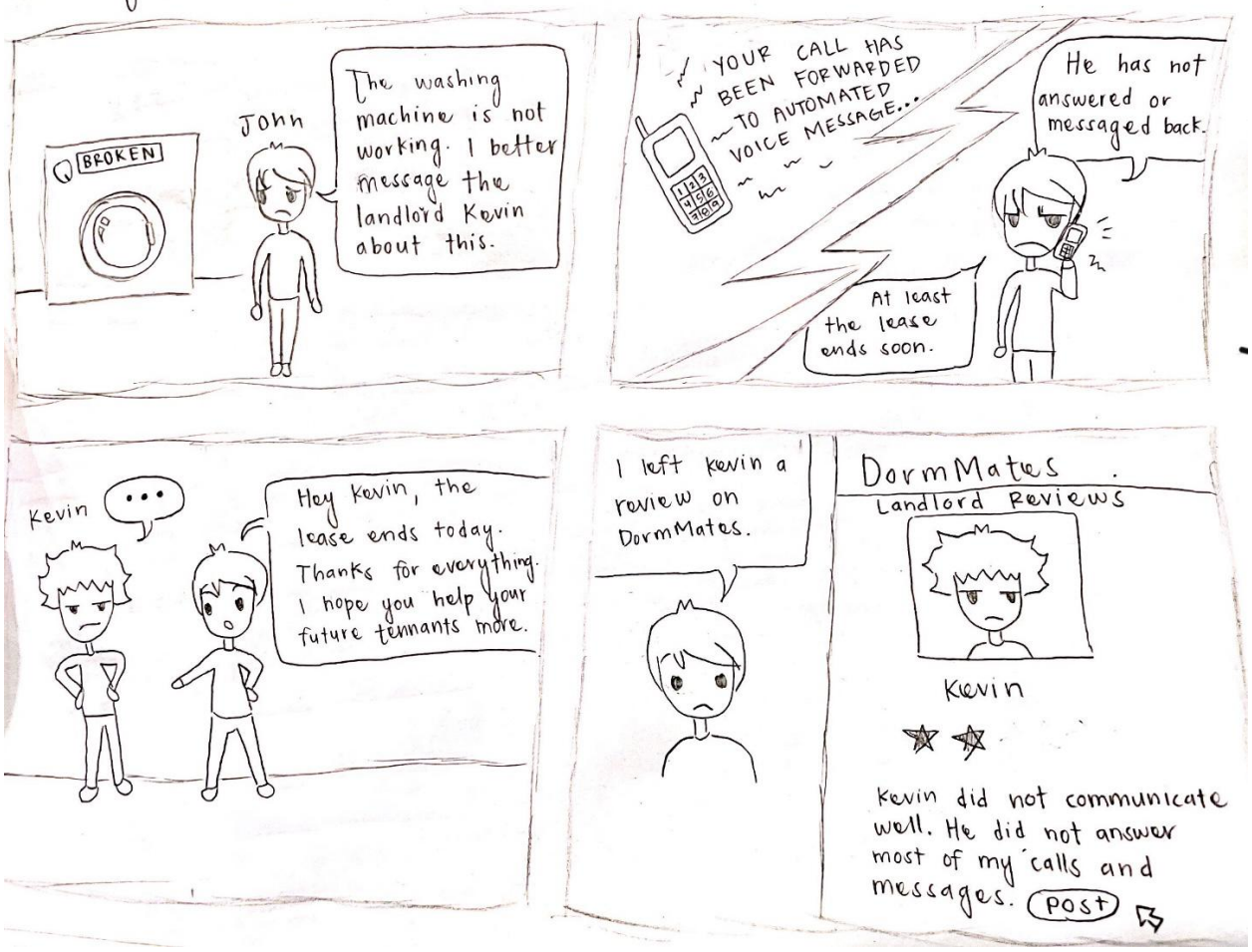
[DormMatesWireframe]

Storyboards

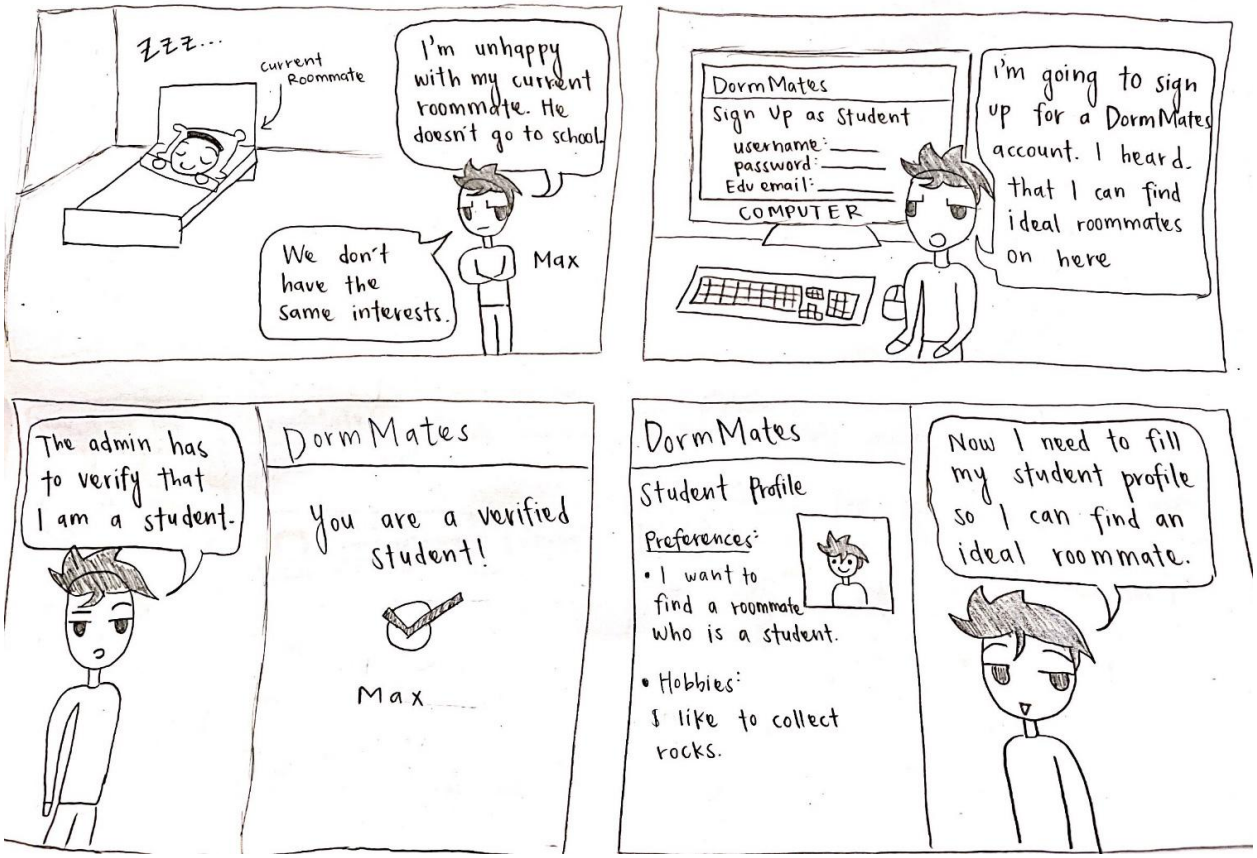


Landlord User Fails to Respond

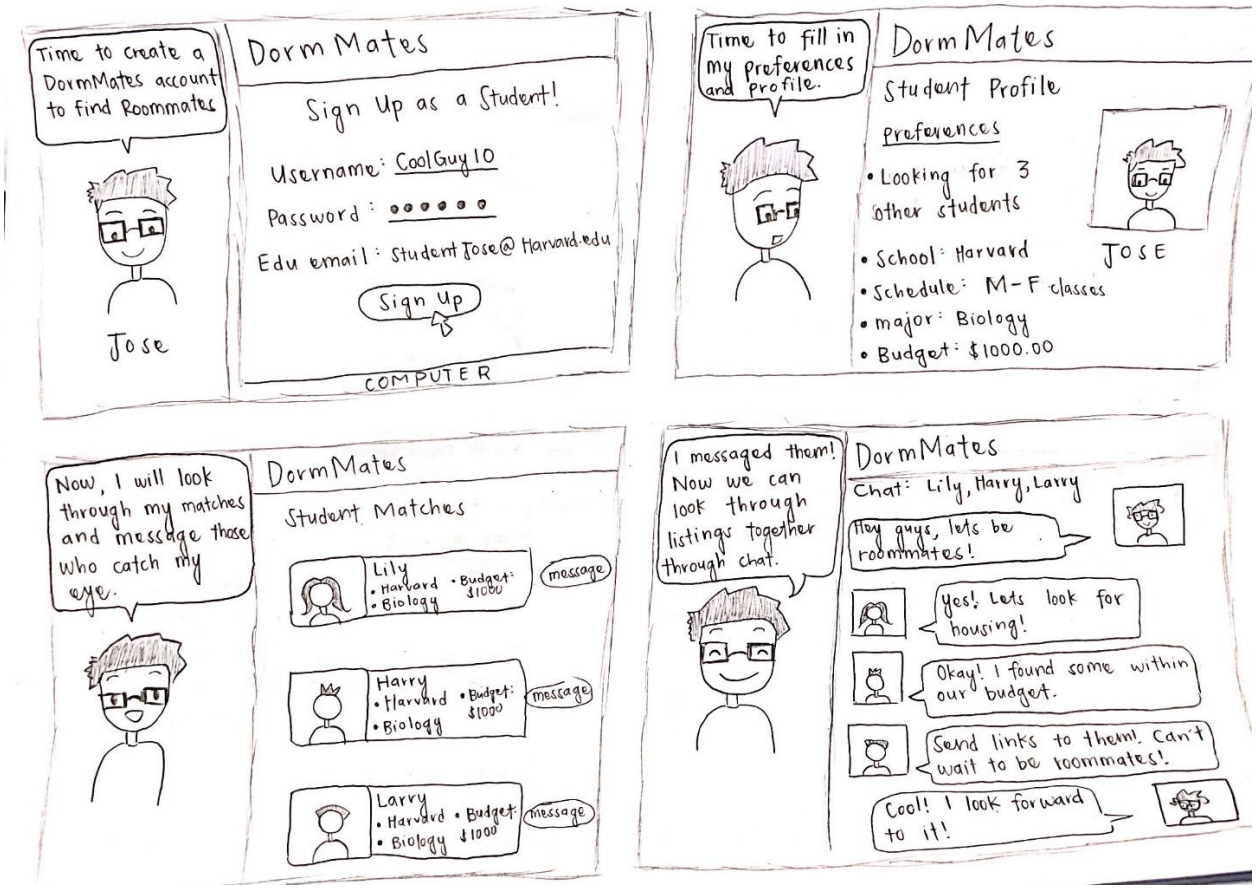
Creating a Review



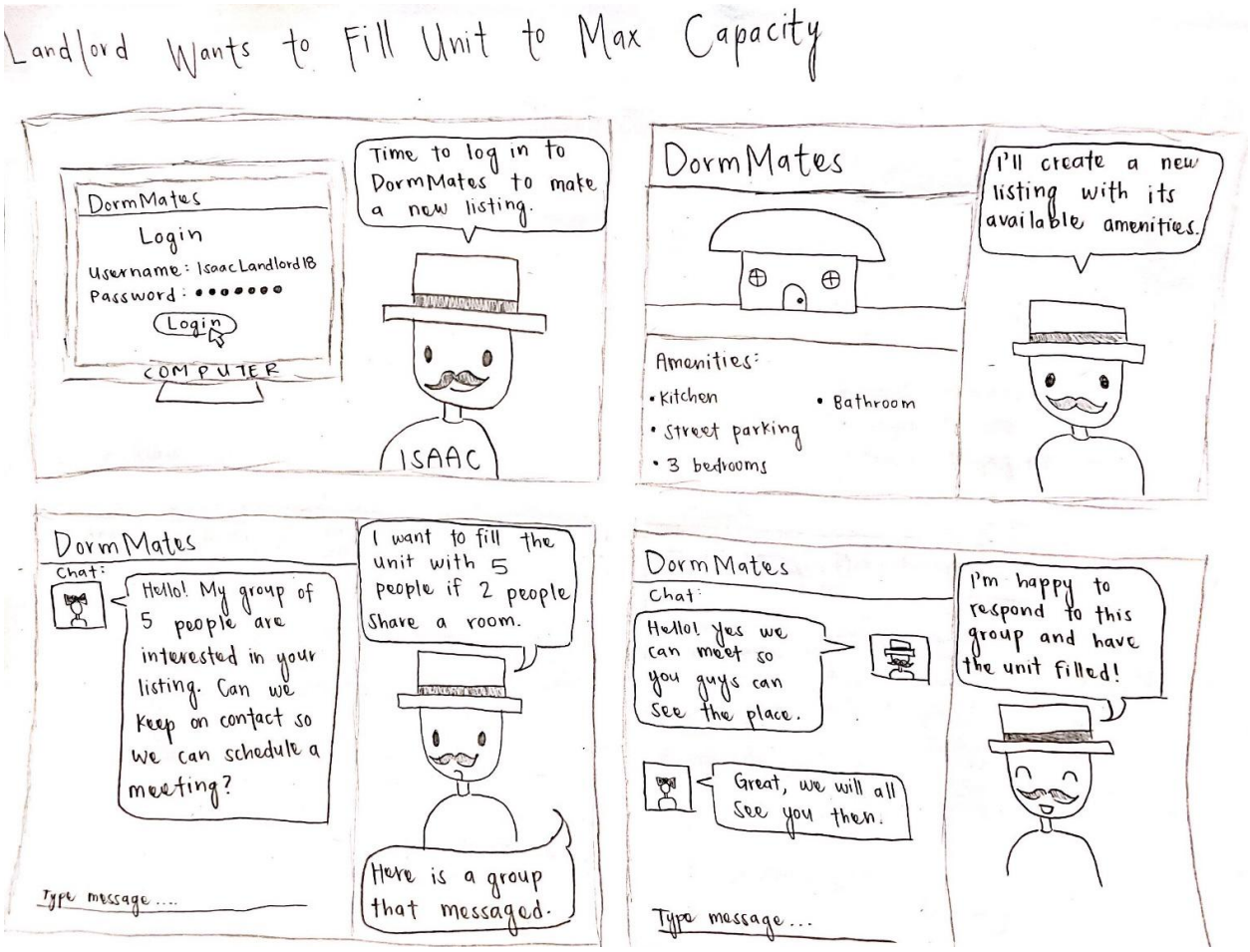
Student Creates a New Account



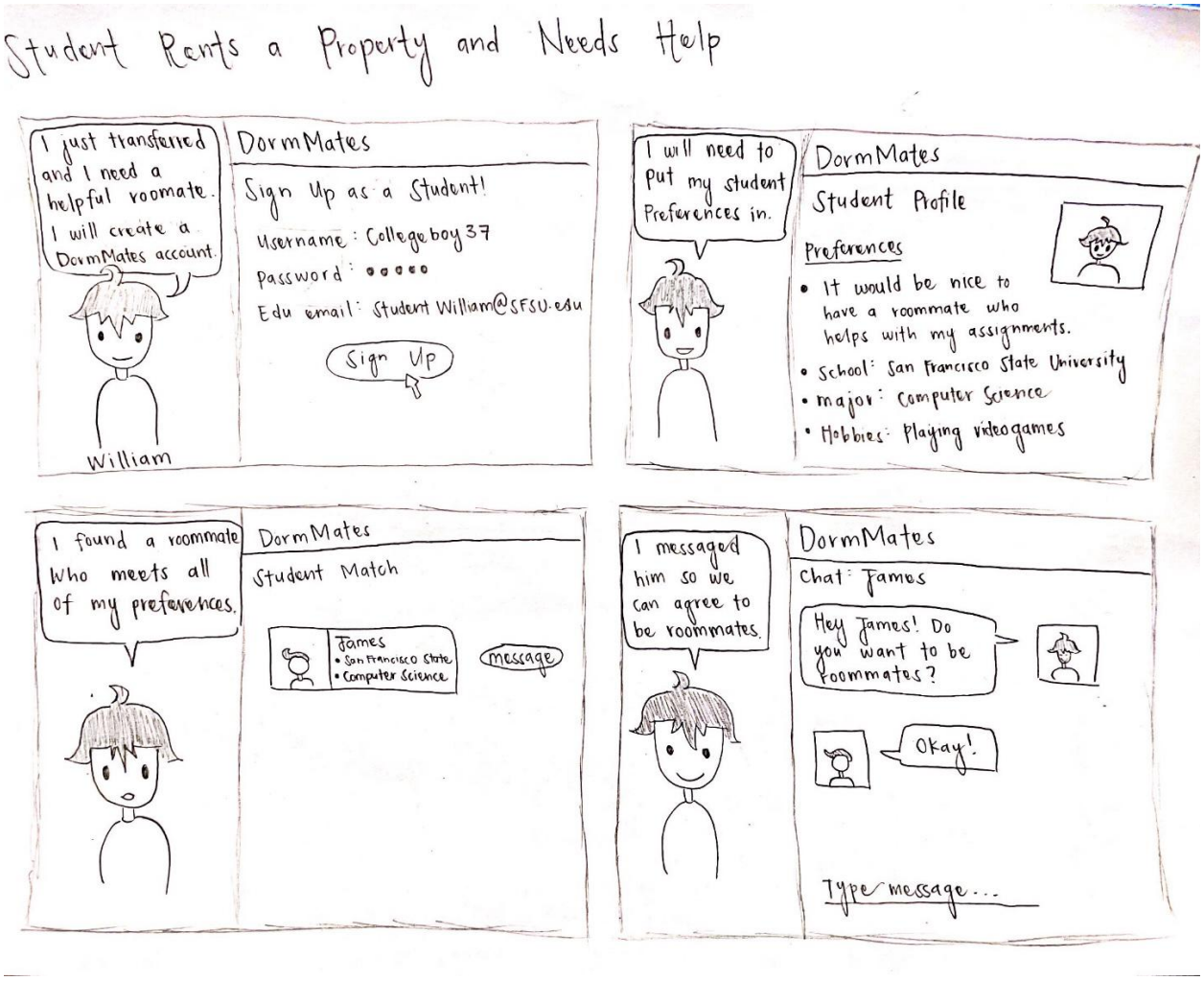
Student Needs Roommates

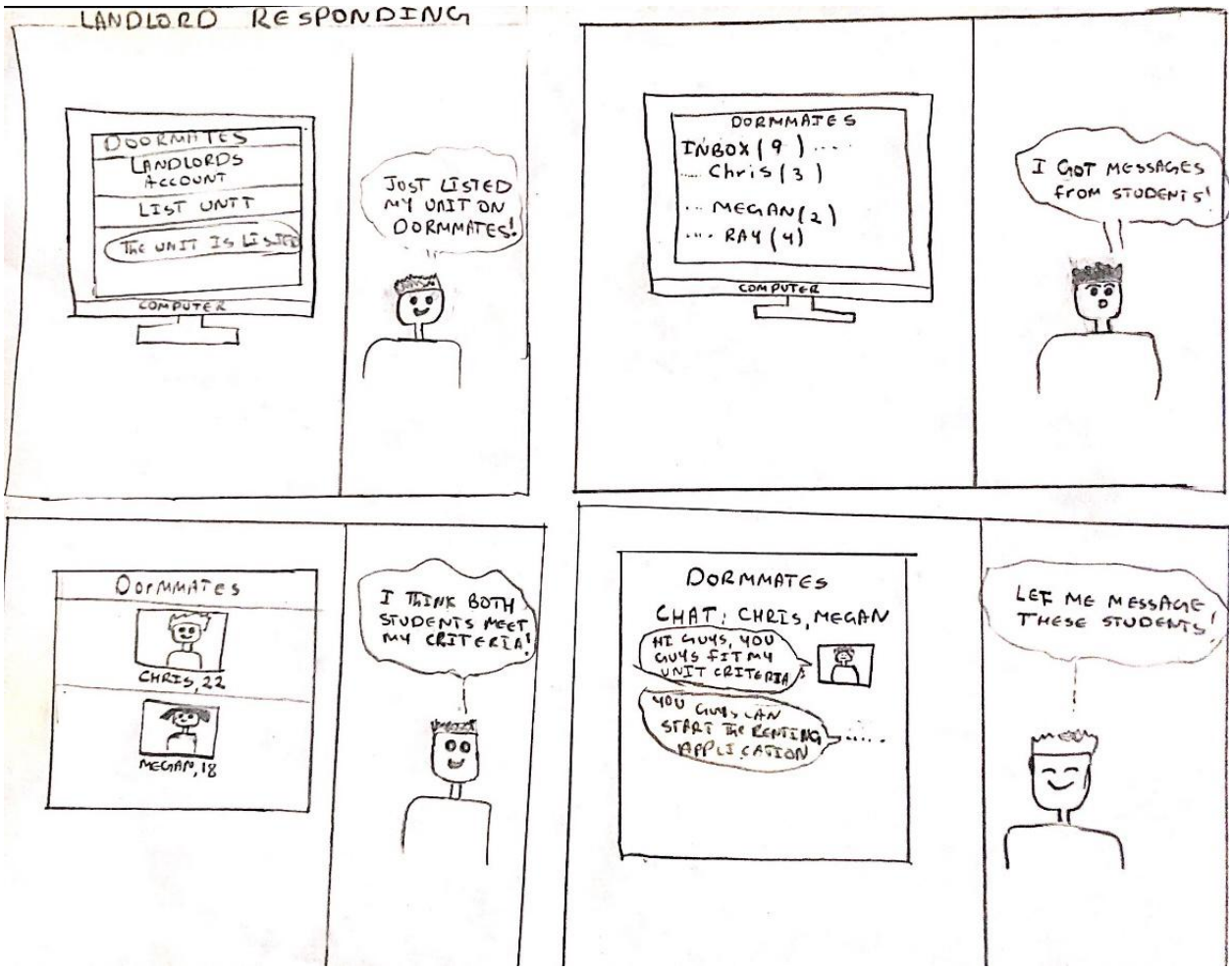


Landlord Wants to Fill Unit to Max Capacity



Student Rents a Property and Needs Help



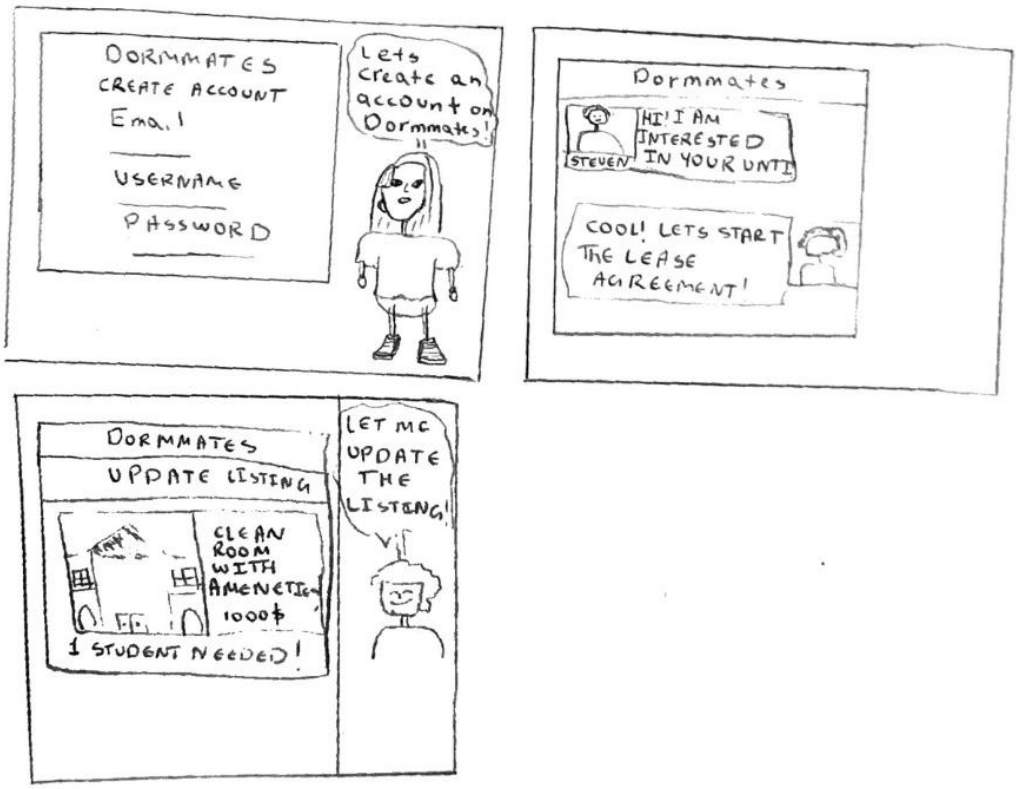

LANDLORD RESPONDING
DORMMATES
LANDLORDS ACCOUNT
LIST UNIT
THE UNIT IS LISTED
COMPUTER
JUST LISTED MY UNIT ON DORMMATES!
DORMMATES
INBOX (9)
Chris (3)
MEGAN (2)
RAY (4)
COMPUTER
I GOT MESSAGES FROM STUDENTS!
DORMMATES
CHRIS, 22
MEGAN, 18
I THINK BOTH STUDENTS MEET MY CRITERIA!
DORMMATES
CHAT: CHRIS, MEGAN
HI GUYS, YOU GUYS FIT MY UNIT CRITERIA
YOU GUYS CAN START THE RENTING APPLICATION
LET ME MESSAGE THESE STUDENTS!

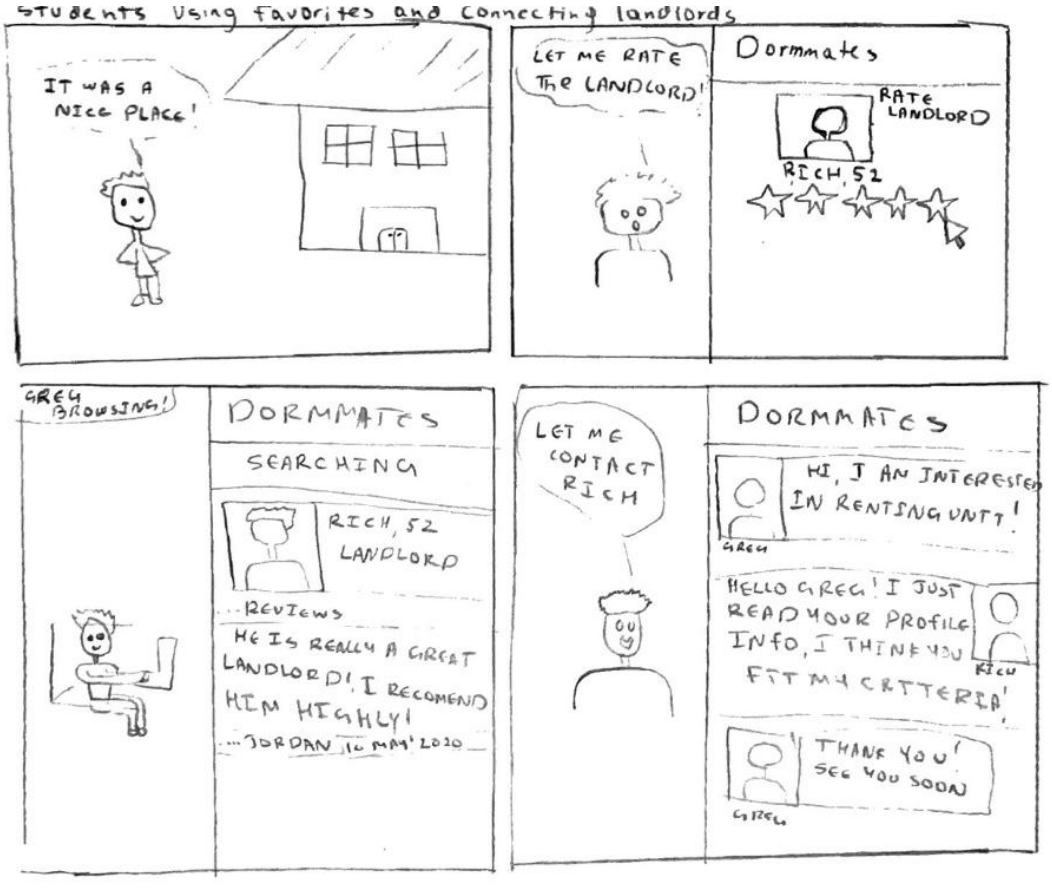

LANDLORD MANAGING MULTIPLE UNTTS
LET ME CREATE LISTINGS ON DORMMATES
DORMMATES
LANDLORD ACCOUNT
LIST UNIT
MEGAN, 45
YOU HAVE CREATED THE LISTINGS!
NICE! I CREATED MULTIPLE LISTINGS!

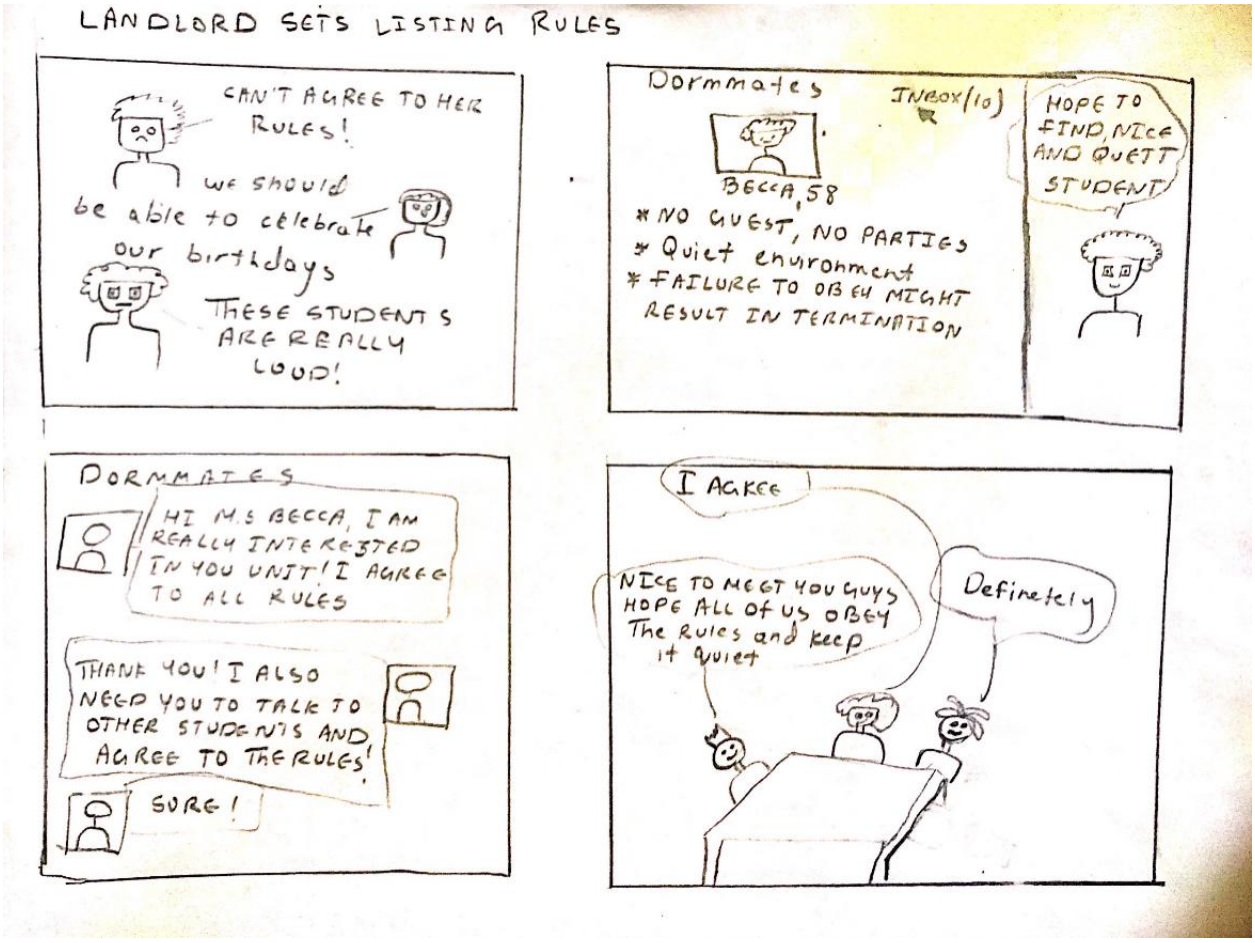
STUDENT CHANGING TEIR LOCATION


STUDENT SEARCHING FOR ROOM
DORMMATES
SIGN UP
EMAIL
USERNAME
PASSWORD
SIGN UP
JUST SIGNED UP
DORMMATES
MENU
ZIPCODE
PRICE
FILTERS
STONERIDGE
$1000
VINTAGE
MAP
DORMMATES
CHATS
HI, DO YOU GO TO UCSD?
10:AM
HELLO, YES!
10:20 AM
NICE TO MEET YOU!
10:20AM
NICE PLATFORM!

LANDLORD POSTING A LISTING AND UPDATING CAPACITY.




Students Using favorites and connecting landlords
IT WAS A NICE PLACE!
LET ME RATE THE LANDLORD!
Dormmates
RATE LANDLORD
RICH, 52
GREG BROWSING!
DORMMATES
SEARCHING
RICH, 52 LANDLORD
...REVIEWS
HE IS REALLY A GREAT LANDLORD! I RECOMEND HIM HIGHLY!
...JORDAN, 16 MAY 2020
LET ME CONTACT RICH
DORMMATES
HI, I AN INTERESTED IN RENTING UNIT!
GREG
HELLO GREG! I JUST READ YOUR PROFILE INFO, I THINK YOU FIT MY CRITTERIA!
RICH
THANK YOU! SEE YOU SOON
GREG


LANDLORD SETS LISTING RULES
CAN'T AGREE TO HER RULES!
we should be able to celebrate our birthdays
THESE STUDENTS ARE REALLY LOUD!
Dormmates
INBOX(10)
BECCA, 58
* NO GUEST, NO PARTIES
* Quiet environment
* FAILURE TO OBEY MIGHT RESULT IN TERMINATION
HOPE TO FIND, NICE AND QUETT STUDENT
DORMMATES
HI MS BECCA, I AM REALLY INTERESTED IN YOU UNIT! I AGREE TO ALL RULES
THANK YOU! I ALSO NEED YOU TO TALK TO OTHER STUDENTS AND AGREE TO THE RULES!
SURE!
I AGREE
NICE TO MEET YOU GUYS HOPE ALL OF US OBEY The Rules and keep it Quiet
Definetely

High Level Database Architecture and Organization

Business Rules

- **Registered User:**
  - A registered user shall be able to log into one account.
  - A registered user shall have zero to many messages.
  - A registered user shall have zero to many sessions.
  - A registered user shall have zero to many reports.
- **Landlords:**
  - A landlord shall have zero to many ratings.
  - A landlord shall have zero to many listings.
  - A landlord shall have zero to many groups.
- **Student:**
  - A student shall have zero to many favorites.
  - A student shall have zero to many groups.
  - A student shall have one institution.
- **Messages:**
  - A message must have one registered user
  - A message must have one group.
- **Groups:**
  - A group shall contain two or more students
  - A group shall contain zero to one landlord
- **Listing:**
  - A listing shall have one landlord.
  - A listing shall have zero to many leases.
  - A listing shall have zero to many photos.
  - A listing shall have one amenities.
  - A listing shall have zero to many leases.
- **Photos:**
  - A photo shall have one listing.
- **Leases:**
  - A lease shall have one listing.
  - A lease shall have one student.
- **Favorites:**
  - A favorite shall have one listing.
  - A favorite shall have one student.
- **Reports:**
  - A report shall have two registered users.

Entities

- **Registered User (Strong)**
  - registerd_id: key, alphanumeric
  - EmailAddress: alphanumeric
  - Password: key, numeric
  - Username: alphanumeric
  - Name: multivalued, alphanumeric
  - Birthdate: numeric
  - Gender: alphanumeric
  - Photo: Image BLOB
  - Last seen: numeric
  - Verified: boolean

- **Admin (Weak)**
  - id: key, alphanumeric

- **Student user (Weak)**
  - Id: key, alphanumeric
  - UserID: key, alphanumeric
  - Major: alphanumeric
  - Personality: alphanumeric
  - Schedule: alphanumeric
  - Institution: alphanumeric
  - Hobby1: alphanumeric
  - Hobby2: alphanumeric

- **Landlord user (Weak)**
  - Id: key, alphanumeric
  - Rating: numeric
  - UserID: weak key, numeric
  - NumOfRatings: numeric

- **Institution (Strong)**
  - Id: key , numeric
  - Name: alphanumeric
  - Domain: alphanumeric
  - Address: alphanumeric

- **Listing (Weak)**
  - Id: key, alphanumeric
  - LandlordID: weak key, alphanumeric,
  - Location: alphanumeric
  - Description: alphanumeric
  - Availability: alphanumeric
  - Amenities: alphanumeric

- **Amenities (Weak)**
    - Id: key, alphanumeric
    - ListingId: weak key, alphanumeric
    - Description: alphanumeric

- **Report (Weak)**
    - ReportId: key, alphanumeric
    - ReporterId: weak key, alphanumeric
    - ReportedId: weak key, alphanumeric
    - Message: alphanumeric
    - Timestamp: numeric

- **Favorite (Weak)**
    - ListingId: key, alphanumeric
    - UserId: weak key, alphanumeric

- **Lease (strong)**
    - LeaseId: key, alphanumeric
    - ListingId: weak key, alphanumeric
    - Contract: alphanumeric
    - Range: composite, numeric

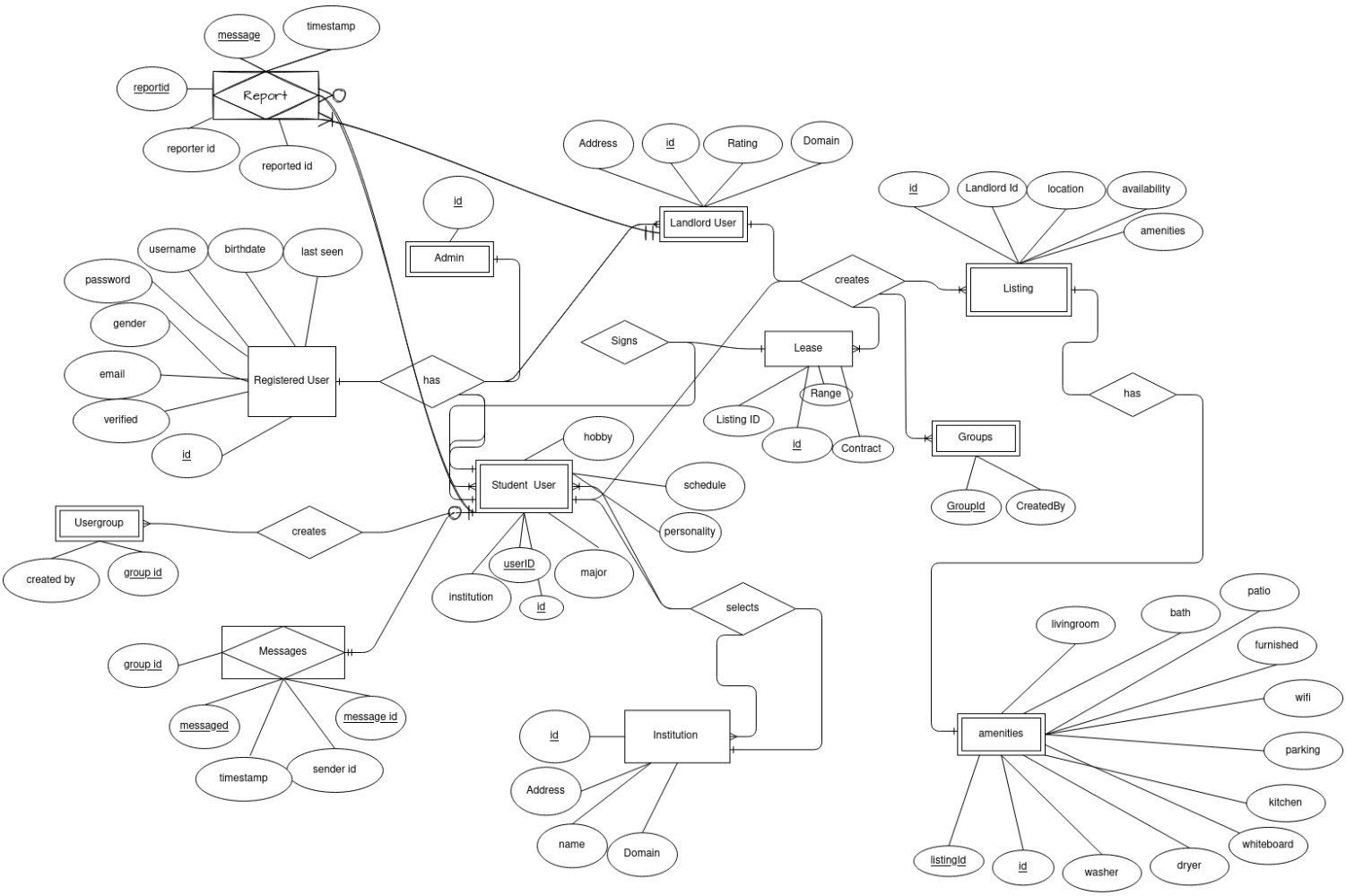
- **Groups (Strong)**
    - GroupId: key ,alphanumeric,
    - CreatedBy: weak key, alphanumeric

- **UserGroup (Weak)**
    - GroupId: key, alphanumeric
    - UserId: weak key, alphanumeric

- **Messages (Weak)**
    - MessageId: key, alphanumeric
    - GroupId: key, alphanumeric
    - SenderId: weak key, alphanumeric
    - Message: alphanumeric
    - Timestamp: numeric

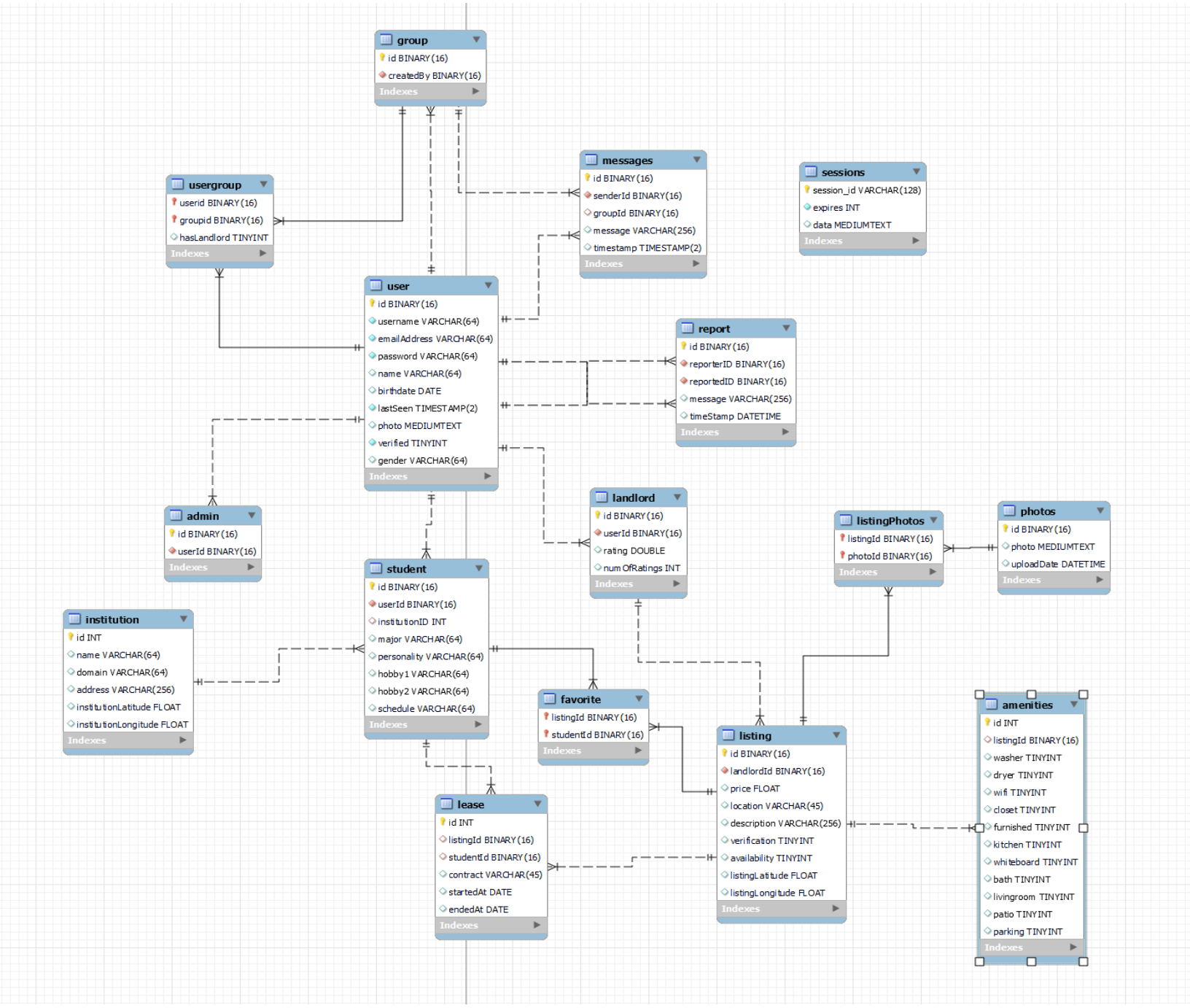
ERD Diagram

Link to diagram (draw.io)



Database Model

DB SQL (pastebin.com)



DBMS:
For the Database, we will be using MySQL because it is a relational database (requirement) and it is what we are most familiar with as a team.

Media Storage:
We will be storing all media in the database as BLOBs. We plan to use Multer to accomplish this.

Search Filer / Architecture and implementation:
For the search filters, we will be using SELECT statements with things like WHERE and LIKE clauses.

High Level APIs and Main Algorithms

- **Authentication API**
  - **Registration**
    - Post Request: When the backend receives a post request when a user signs up for an account using an email, username, and password. If all the inputs are valid then a successfully registered message is shown. Otherwise a responding failure to register account message is displayed.
  - **Login**
    - Post Request: The backend receives a post request when a user attempts to login using either a combination of their username and password or their email and password. The backend responds to the request by validating the login by checking if the user exists in the database, their email is correct, their username is correct, and/or their password is correct. When either the user doesn't exist or the login credentials are incorrect then the backend responds with an error message.

- **Chat API**
  - **Send**
    - Post Request: The backend receives a post request when a user begins a chat with one or more users. The backend checks for an id that matches with the group. If there are no groups that match an id then the chat is not sent.
  - **Read**
    - Get Request: When a user attempts to open a chat the backend receives a get request. The backend checks for an id that matches with the group and displays an array of data when found. If a group id is not found then an error is to be displayed.

- **Search API**
    - **Filter Listing Search**
        - Get Request: When a user views listing the backend receives the get request. The backend checks to see which listing id matches the user's requirements based on query parameters and displays all listings that meet the criteria. If no listing that meets the user's requirements is found then the backend responds with a message stating that zero listings were found that match the requirements.
    - **Filter Roommate Search**
        - Get Request: When a student user views the list of all available roommates near them the backend receives the get request. The backend searches for student ids that meet the student user's requirements based on query parameters and display roommate options that meet them. If no roommate option meets the user's preferences then the backend sends a message to be displayed stating no roommates meet the requirements.
    - **Find Institutions**
        - Get Request: When a user searches for institutions the backend receives a get request. The backend searches for attributes that are similar to the user's input and displays them. If no institutions are similar to the user's input then a message displays stating no institution can be found.

- **Rating API**
    - **Landlord Rating**
        - Post Request: A user can rate a landlord that they have rented from. The backend saves the input and notify the user that they have successfully rated the landlord. If the backend fails to do so the user receives a notification through an error message.
        - Get Request: When a user views a landlord, the user sees the rating of the landlord. The backend receives a get request with the landlord's id. If the backend fails to find an id then the backend responds with an error message.

- **Listing API**
    - **Add a listing**
        - Post Request: When a landlord user attempts to upload a listing the backend receives a post request. When all data fields are confirmed to be valid the backend saves the listing and displays that the listing is successfully added. If any of the data fields are invalid then the backend displays an error message stating that one or more fields are invalid.
    - **Edit a listing**
        - Put Request: A landlord user has the ability to edit listings they already uploaded. The backend receives the post request and updates the data fields that the landlord user has changed by first checking if the listing exists in the database by searching for the listing id.
    - **Delete a listing**
        - Delete Request: When a landlord user wants to delete one of the listings that they have on the service the backend receives a delete request. The backend searches for a matching listing id and removes the matching listing from the database. If no listing id is found then the backend displays an error message stating the listing could not be deleted.

- **Favorites API**
    - **Favoriting a listing**
        - Post Request: When a student favorites a listing the backend searches in the database for the listings id. If the backend finds the listing id it saves the listing id and pairs it with the student user id and displays a messaging stating that the listing was successfully favorited. Otherwise the backend displays a message stating that the listing was not able to be favorited.
    - **Favorites List**
        - Get Request: A student user viewing their favorites list sends a get request to the backend. The backend searches for listing ids and matches them with the student user's id and displays all listings that match. If no matches are made then the backend sends a message that displays no listings are favorited.

- **Lease API**
    - **Create a lease**
        - Post Request: A landlord adds a lease to a listing. The backend shall match the lease with the listing and save them in the database and display a message stating that the lease was successfully uploaded. If this fails then the backend sends a message stating that the lease was not uploaded.
    - **Sign a lease**
        - Post Request: A student user shall be able to sign a lease on listing. The backend then matches the lease with the student id and saves them within the database and displays a message stating that the lease was signed. If an error occurs then the backend sends a message stating that the lease was unable to be signed.
    - **View a lease**
        - Get Request: When a user views a listing a lease is displayed within that listing. The backend searches the database for a listing id to match with a lease id to display the lease.

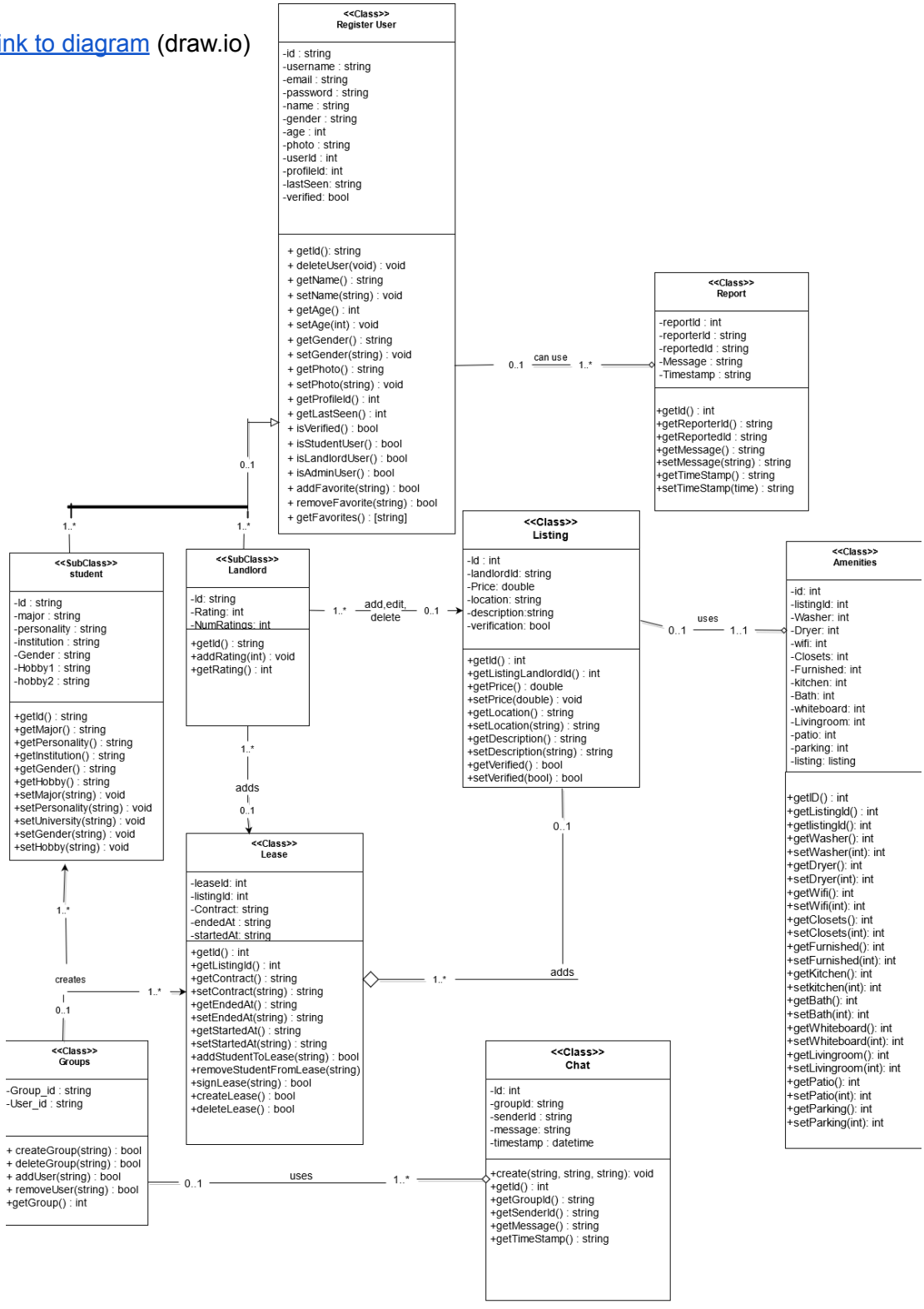
- **Report API**
    - **Create a report**
        - Post Request: When a user reports a listing or another user the backend receives a post request. The backend searches for the id of either the listing or user being reported. If found then the backend sends a message stating the report was received. Otherwise the backend sends an error message.

- **Rating Algorithm**
    - Our service shall implement a rating algorithm that for when users attempt to contact a landlord and the landlord fails to reach them after a certain amount of time passes then their ratings will drop.

No new software has been added to the list of softwares our team has decided on using.

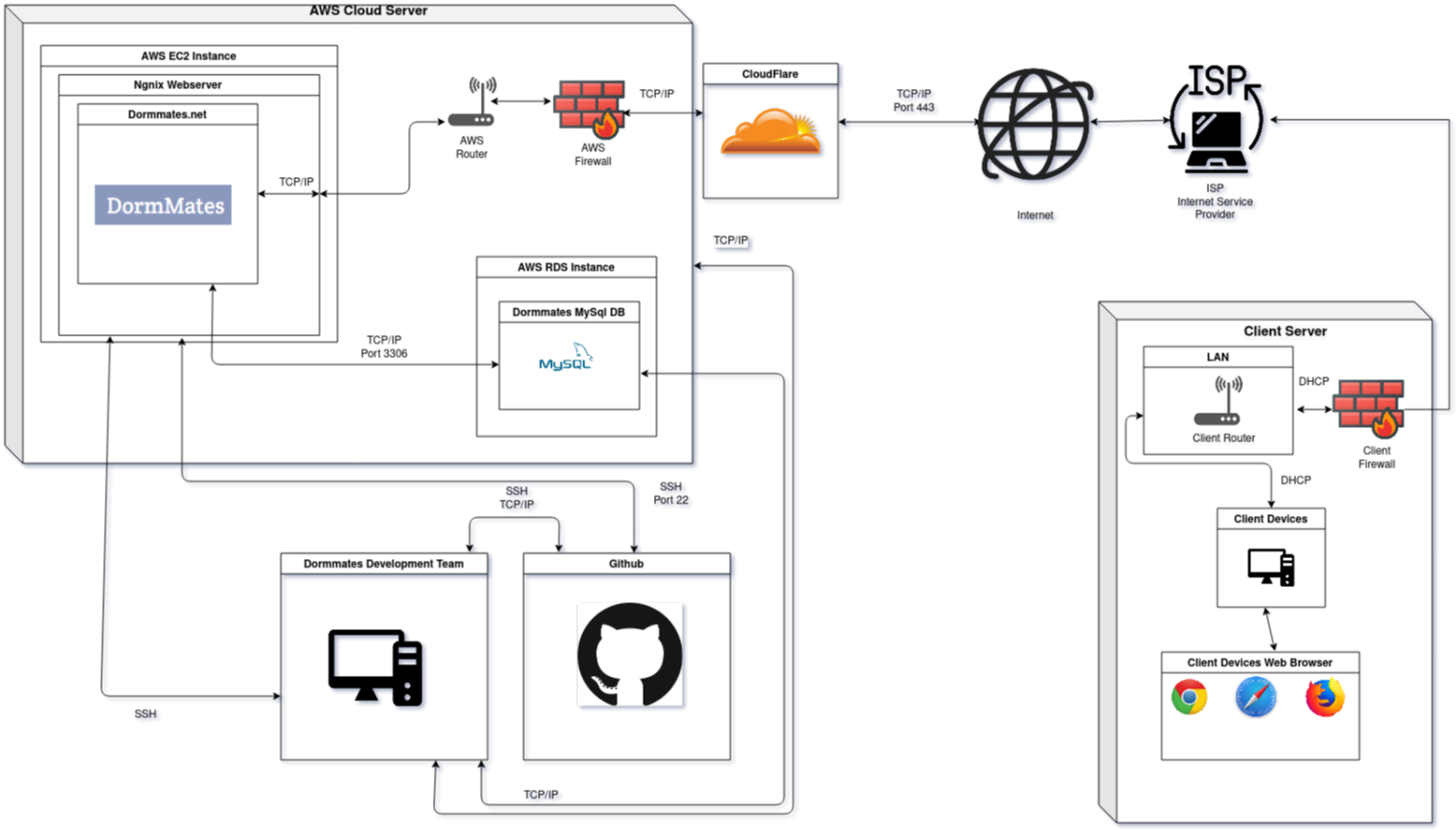
High Level UML Diagrams

[Direct link to diagram](#) (draw.io)

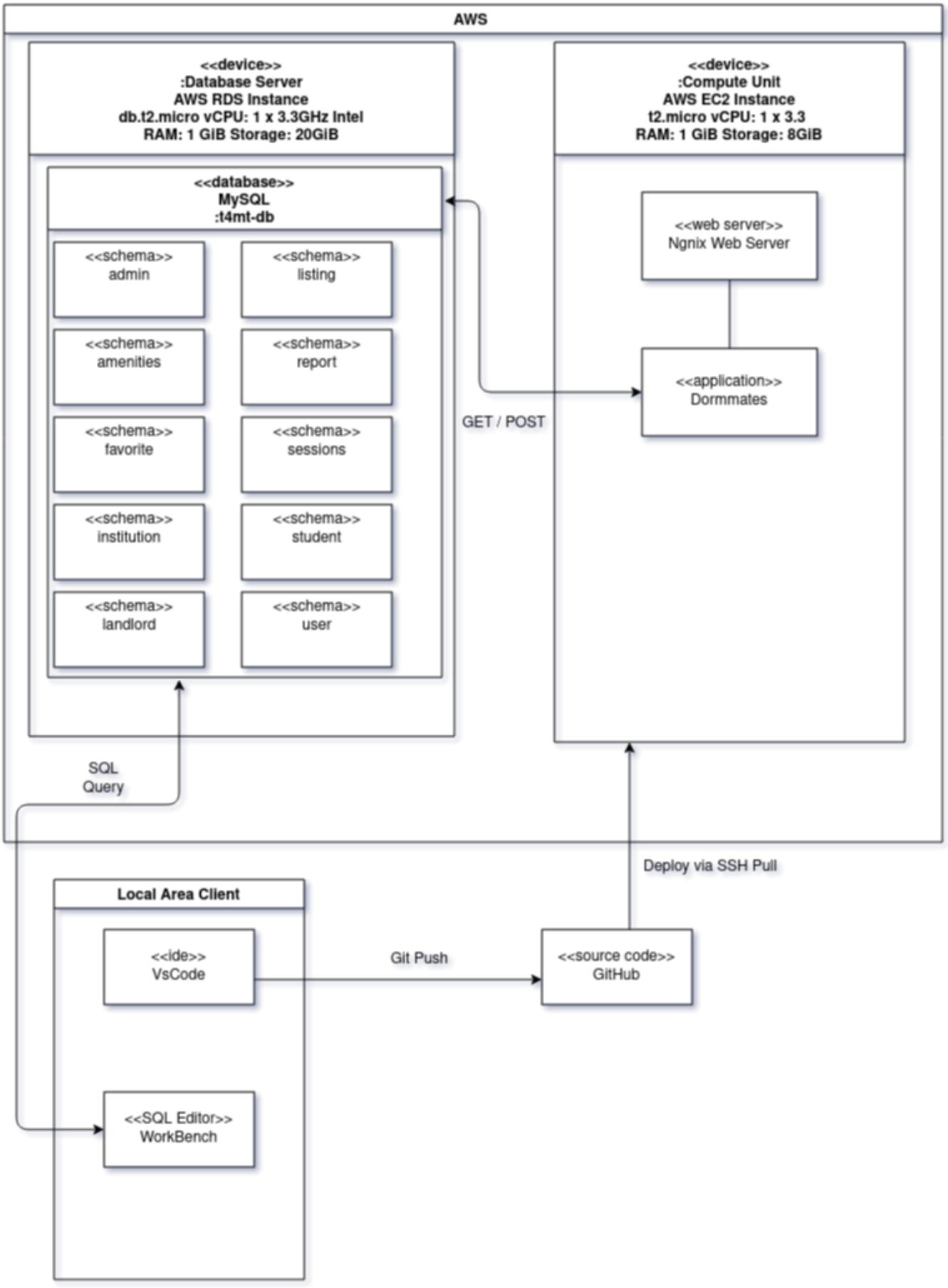


High Level Application Network and Deployment Diagrams

Network Diagram



Deployment Diagram



Key Risks

Skills

- **Writing**
  - Brainstorming the ideas in a document.
  - Needing more ideas to write in a document.
  - Don't be afraid to write anything; mistakes are encouraged and we should address them as a team.
  - Asking questions about things that may not be clear.
- **Gathering information**
  - Refer to the professor's milestone documents.
  - Asking the discord to clarify if you are unsure about anything.
- **Professor expectations:**
  - The team will communicate with the professor via Discord and email to clarify on expectations.

Schedule

- **Do the work before the agenda's due time.**
  - Individuals should not wait until the last second to complete agenda tasks.
  - When the agenda is posted, individuals should mark it seen with the SWE stickers
  - When the prior tasks are done in the agenda, the individuals should mark it seen with the 100% sticker.
- **Accommodating for every team members' schedule (classes, personal)**
  - The team lead will be more aware of everyone's schedules and conduct meetings with them in mind.
- **Keeping track of tasks on Jira:**
  - The team lead will monitor Jira frequently and check on progress of tasks.
- **Timely meetings:**
  - Start the meeting within 5 minutes of the start time, end within 15 minutes of estimated time
- **Keep meetings productive:**
  - The agenda will include an estimated timeline for the meeting along with the listed tasks.
  - Agenda should include a max allocated time for each meeting.
  - The team lead should give out breaks during the meeting.
  - The team lead should ensure that all meetings are on task.

Technical

- **Organization:**
  - Milestone editor will be responsible for formatting all content.
  - The github master will ensure that all coding guidelines are being followed.
- **Using technologies.**
  - Making sure everyone is comfortable using Mysql
    - The backend team will individually review/refresh mysQL before starting the first task.
  - Making sure everyone understands the framework for frontend.
  - Making sure all backend teammates are comfortable using javascript.
  - Making sure all backend teammates are comfortable with Express.
  - Making sure everyone knows how to use the discord bot.

    - Make an effort to read any documentation for said technology.

- **Knowledge of frameworks**
    - Reading documentation for frameworks
    - Looking at examples of frameworks being used
    - Asking for help or for clarity

Teamwork

- **Making sure that everyone is contributing to the project:**
    - Alerting the team lead if someone isn't contributing as much as they should.
    - The team lead will be more aware of individual contributions and address the lack of contributions swiftly.
    - Make sure every team member is involved and happy with their contributions
- **All members are attending team meetings:**
    - If they can't attend meetings, they are watching the meeting recordings.
        - Any team member who must watch a recorded meeting should create a writeup commenting on what they missed to show that they are on the same page.
    - The team lead will send out reminders for all meetings.
- **Productivity:**
    - The team lead will create internal deadlines and ensure that they are being met.
    - Ensuring deadlines and requirements are made clear
- **Ensuring all members agree on deadlines**
    - Clear and concise tasks on Jira with internal deadlines.
    - Ensuring that the majority decision policy is always in effect.
- **All members completing their task on time:**
    - The team lead will monitor tasks on Jira to ensure progress is being made.
    - Team members will give notice of any issues they are having or complications.

Legal/Content

- **Copyrighted Content**
    - All teammates must verify any content used on the site is not copyrighted
    - Ensuring we give credit to artwork.
    - Ensuring that all software used in our project is open source/has a license that allows us to use it freely.
- **Ensuring all users accept and respect our terms of use:**
    - Users will not be able to register without accepting the terms and use
- **Ensuring users should be 18 and over:**
    - We'll require all unregistered users to confirm that they are 18 or older when creating a new account.

Project Management

For project management we are using Jira by Atlassian.

In milestone 2 and all future milestones, the team lead works with the frontend and backend leads to dissect the milestone document and break down high level tasks into many sub tasks that are then assigned to individual members through Jira.

Each sub task has a due date, description, and child tasks. The due date is when the task and all of its child tasks are due, the description is a high level description of the task, and the child tasks are itemized tasks that break the sub task down into checkpoints that should be completed in the order they are listed.

Detailed List of Contributions

Andrei

- Managed the project by creating tasks on Jira and dispatching them to the team.
- Managed the project by keeping up with individual members of the team to track progress and give feedback.
- Met with the frontend and backend team frequently to give advice and feedback on work being done.
- Hosted Zoom meeting where the team worked on addressing Milestone 1 feedback.
- Hosted Zoom meetings where the team worked on Milestone 2 tasks.
- Hosted meetings over Discord with backend and frontend teams to brainstorm implementation.
- Hosted pair programming sessions with the team for setting up the project development environment.
- Hosted pair programming session with the backend team for creating the backend institution search endpoint.
- Hosted pair programming session with the frontend team for connecting the frontend and the backend.

Jonathan

- Attended all team meetings.
- Communicated with the team through Discord often.
- Hosted meetings over Discord with the backend team to brainstorm backend implementation.
- Hosted pair programming session for creating backend user model, controller, route, and endpoint.
- Helped brainstorm the content for the Data Definitions task.
- Helped brainstorm the content for the Prioritized Functional Requirements task.
- Helped brainstorm the content for the Key Risks task.
- Implemented the User API.
- Worked on the High Level Database Architecture and Organization task.
- Attended all backend pair programming sessions with the team.

Meeka

- Attended all team meetings.
- Communicated with the team through Discord often.
- Hosted meetings over Discord with the frontend team to brainstorm frontend implementation.
- Helped brainstorm the content for the Data Definitions task.
- Helped brainstorm the content for the Prioritized Functional Requirements task.

- Helped brainstorm the content for the Key Risks task.
- Worked on the Use Case Mockup / Storyboard milestone task.
- Created Wireframes for the vertical prototype.
- Created Pug views for wireframes they created.
- Created CSS styles for the frontend.
- Attended all frontend pair programming sessions with the team.

Jimmy

- Attended all team meetings.
- Communicated with the team through Discord often.
- Helped brainstorm the content for the Data Definitions task.
- Helped brainstorm the content for the Prioritized Functional Requirements task.
- Helped brainstorm the content for the Key Risks task.
- Worked on documenting what API routes the backend team needed to implement.
- Worked on the High Level UML Diagrams task.
- Worked on the High Level Application Networks and Deployment Diagrams task.
- Attended all backend pair programming sessions with the team.

Alexandre

- Attended all team meetings.
- Communicated with the team through Discord often.
- Helped brainstorm the content for the Data Definitions task.
- Helped brainstorm the content for the Prioritized Functional Requirements task.
- Helped brainstorm the content for the Key Risks task.
- Worked on the High Level APIs and Main Algorithms task.
- Worked on the High Level Application Networks and Deployment Diagrams task.
- Created documentation for the backend code.
- Created documentation for the deployment process of the application.
- Attended all backend pair programming sessions with the team.

Sayed

- Attended all team meetings.
- Communicated with the team through Discord often.
- Helped brainstorm the content for the Data Definitions task.
- Helped brainstorm the content for the Prioritized Functional Requirements task.
- Helped brainstorm the content for the Key Risks task.
- Worked on the Use Case Mockup / Storyboard milestone task.
- Created Wireframes for the vertical prototype.
- Created Pug views for wireframes they created.
- Created CSS styles for the frontend.
- Attended all frontend pair programming sessions with the team.

Ahad

- Attended all team meetings.
- Helped brainstorm the content for the Data Definitions task.
- Helped brainstorm the content for the Prioritized Functional Requirements task.
- Helped brainstorm the content for the Key Risks task.
- Compiled a list of all colleges in Northern California and inserted them into our database.
- Worked on the High Level Application Networks and Deployment Diagrams task.
- Attended some backend pair programming sessions with the team.

SW Engineering CSC 648 Summer 2021

DormMates

Milestone 3 • Version 2 • 30 July 2021

Team 01

Team Lead and Github master	Andrei Georgescu	ageorgescu@mail.sfsu.edu
Frontend Lead	Meeka Cayabyab	
Backend & Database Lead	Jonathan McGrath	
Engineer	Alexandre Ruffo	
Engineer	Jimmy Yeung	
Engineer	Sayed Hamid	
Engineer	Ahad Zafar	

History Table

Version	Date	Notes
M3V2	08/01/2021	Addressed wireframe and diagrams feedback.
M3V1	07/22/2021	Initial submission
M2V2	07/19/2021	Addressed the vertical prototype, functional requirements, and diagrams feedback.
M2V1	07/08/2021	Initial submission
M1V2	07/01/2021	Added database master role to the title page, addressed use cases feedback, addressed functional requirements feedback, and addressed competitive analysis feedback.
M1V1	06/22/2021	Initial submission

Table of Contents

Table of Contents **3**

Data Definitions V3 **4**

Functional Requirements V3 **7**
- Priority 1 - Required 7
- Priority 2 - Desired 8
- Priority 3 - Opportunistic 9

Wireframes **10**
- Use Case 1: Creating a Landlord Account and Posting a Listing 10
- Use Case 4: Student Creates a New Account 11
- Use Case 5: Student Needs Roommates 12
- All Wireframes 13

High Level Database Architecture and Organization V2 **14**
- Database Model 14

High Level Diagrams V2 **15**
- UML Diagram 15
- Network Diagram 16
- Deployment Diagram 17

Detailed List of Contributions **18**

Data Definitions V3

- **General User**: A user who can view and access the website limited to the website's features. A general user must **register** by creating a student or landlord account to gain more privileges.
    - **Registration**: A general user has the option to create an account.
    - A general user shall be able to view limited listings based on an institution.
    - A general user shall be able to view the informational pages regarding the service.

- **Registered User**: A user who has access to the website's core features.
    - **Account Contains:** All registered user accounts must contain the following information.
        - **Username**: A username is required to create an account.
        - **Email**: A unique email is required to create an account.
        - **Password**: A password that will be hashed for security purposes is required.
        - **Accepted Terms of Use**: Must accept to create an account.
        - **Name:** A registered user must have a first and last name.
        - **Gender:** A registered user must specify a gender.
        - **Age:** A registered user must have an age.
        - **Photo**: A profile shall have a picture of the registered user.
    - Has the ability to log into the service.
        - **Email**: Needs to login with a unique email.
        - **Username**: Needs to login with a unique username.
        - **Password**: Needs to provide a password.
    - Has the ability to communicate with other users.
    - Has the ability to create a student account.
    - Has the ability to create a landlord account
    - **Student:** A user who signed up with a .edu email is denoted a student.
        - Has the ability to create a group.
        - Has the ability to invite other student users to a group.
        - Has the ability to invite landlords to groups.
        - **Profile:** A profile will display information / attributes of a student user.
            - **Institution :** An institution will be part of a profile
            - **Major:** A major will be part of a student profile
            - **Schedule:** Needs to specify if they have a night or day schedule
            - **Hobby**: A hobby will be part of a student profile.
            - **Personalities:** A personality will be part of a student profile.
            - **Social Media:** A link to social media outlets.

    - **Landlord:** A user who signed up with a non .edu email is denoted as a landlord.
        - Has the ability to post new listings.
        - Has the ability to create leases.
        - Has the ability to add students to a lease.

        - **Profile:** A profile will display information / attributes of a landlord.
            - **Listings:** The listings that they have posted
            - **Rating:**A list of ratings given by students.
            - **Student Ratings**: Number of students that have rated the landlord.

- **Admin User**: User that has the ability to moderate the service.
    - Can validate listings and make them public on the platform.
    - Can remove listings from the platform.
    - Can answer reports made by other users on the platform.
    - **Account Contains:** All registered user accounts must contain the following information.
        - **Username**: A username is required for all admin users.
        - **Email**: A unique email is required for all admin users.
        - **Password**: A password that will be hashed is required for all admin users.

- **Listing:** A representation of a housing unit that was posted by a landlord.
    - **Location:** Represents the physical location of a listing.
        - Represented by latitude and longitude coordinates.
    - **Photos:** Represents visual representations of a listing.
    - **Lease:** Represents an agreement between one or more students and a landlord which signifies that a lease between the parties exists.
        - Landlords can create leases for their listings.
        - Students can sign leases.

    - **Amenities:** Represents the key features of a listing that are included.
        - Washer and Dryer
        - Wifi
        - Closets
        - Bathrooms
        - Furnished
        - Whiteboard
        - Living room
        - Kitchen
        - Patio
        - Parking

- **Favorite**: Specific listings that a user is interested in and wants to bookmark.

- **Match:** A representation of two student users who may be compatible roommates.

- **Groups:** A representation of a group of students that are matched and looking for listings together.
    - **Chat:** A way groups can communicate with each other on our platform.

- Student users can share listing links in chat.
- Student users can add a landlord user to a chat.
- Landlord users can create a lease in a chat and invite all student users to sign the lease.

Functional Requirements V3

---

1. Priority 1 - Required

    1.1. Unregistered User

        1.1.1. An unregistered user can create a new student or landlord account.

1.1.2. An unregistered user can view listings near an institution of their choice.
1.1.3. An unregistered user can view the Home page.
1.1.4. An unregistered user can view the Terms of Service page.
1.1.5. An unregistered user can view the FAQ page.
1.1.6. An unregistered user can view the Features pages.
1.1.7. An unregistered user can view the About page.

1.2. Registered User
1.2.1. A registered user should be able to login to with their username and password.
1.2.2. A registered user should be able to logout of their account.
1.2.3. A registered user should be able to submit a forgotten password form.
1.2.4. A registered user should be able to submit a forgotten email form.
1.2.5. A registered user should be able to change their username.
1.2.6. A registered user should be able to change their email address.
1.2.7. A registered user should be able to change their password.

1.3. Students
1.3.1. A student user must have a verified edu email.
1.3.2. A student user must have a completed profile.
1.3.3. A student user shall be able to view the student dashboard.
1.3.4. A student user shall be able to view student user profiles.
1.3.5. A student user shall be able to message another user.
1.3.6. A student user should be able to add other student users to a group.
1.3.7. A student user should be able to add a landlord to a group.
1.3.8. A student user shall be able to search for listings.
1.3.9. A student user shall be able to edit their personality.
1.3.10. A student user should be able to edit their schedule.
1.3.11. A student user should be able to edit their hobbies.
1.3.12. A student user shall be able to filter listings by price.
1.3.13. A student user shall be able to filter listings by amenities.
1.3.14. A student user shall be able to filter listings by distance from university.
1.3.15. A student user shall be able to filter roommate selections by personality.
1.3.16. A student user shall be able to filter roommate selections by major.
1.3.17. A student user shall be able to filter roommate selections by hobbies.
1.3.18. A student user shall be able to filter roommate selections by schedule.
1.3.19. A student user should be able to view the location of a listing on a map.
1.3.20. A student user shall be able to favorite a listing.

1.4. Landlords
1.4.1. A landlord user shall be able to view the landlord dashboard.
1.4.2. A landlord user shall be able to view their own profile.
1.4.3. A landlord user shall be able to post listings.
1.4.4. A landlord user shall be able to edit their listings.
1.4.5. A landlord user shall be able to view student profiles.
1.4.6. A landlord user will be able to respond to student user messages.
1.4.7. A landlord user shall be able to delete their listings.

1.5. Admin
1.5.1. Admin users should be able to approve or deny listings created by landlords.

2. Priority 2 - Desired

2.1. Unregistered User

2.2. Registered User
2.2.1. A registered user should be able to delete a chat.

2.3. Students
2.3.1. A student user should be able to change their university.
2.3.2. A student user shall be able to rate landlords.

2.4. Landlords
2.4.1. A landlord user shall be able view historical listing data.
2.4.2. A landlord user shall be able to repost their listing once expired.

2.5. Admin
2.5.1. Admin users should be able to edit registered user profiles.

3. Priority 3 - Opportunistic

3.1. Unregistered User

3.2. Registered User

3.3. Students

3.3.1. A student user shall be able to report a listing.
3.3.2. A student user shall be able to report another user.

3.4. Landlords
3.4.1. A landlord shall be able to report a listing.
3.4.2. A landlord shall be able to report another user.

3.5. Admin
3.5.1. Admin users should be able to disable non-admin user accounts.
3.5.2. Admin users should be able to remove a listing.
3.5.3. Admin users should be able to view reports.
3.5.4. Admin users should be able to respond to reports.

Wireframes

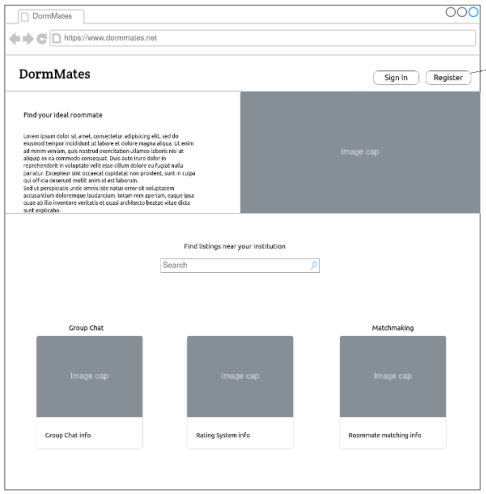
Use Case 1: Creating a Landlord Account and Posting a Listing

1. General user visits the website for the first time with the intention of creating an account.

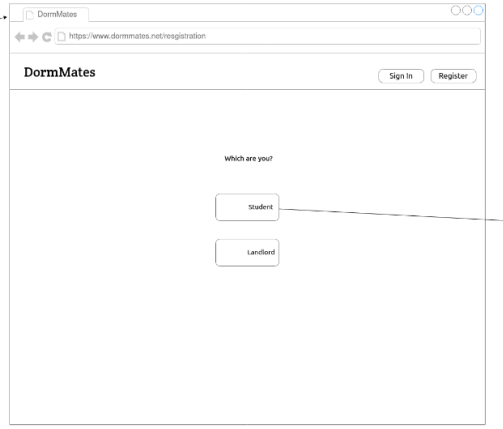
2. User selects the type of account they want to create

Use Case 4: Student Creates a New Account

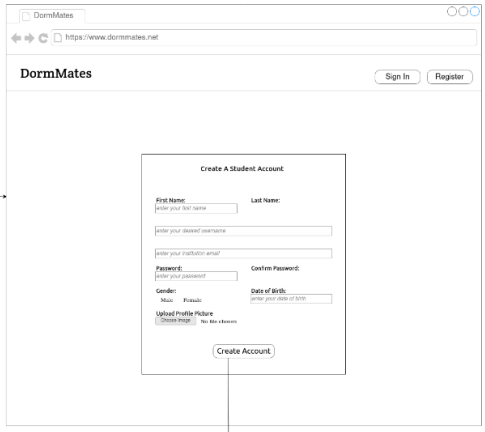
1. General user visits the website for the first time with the intention of creating an account.



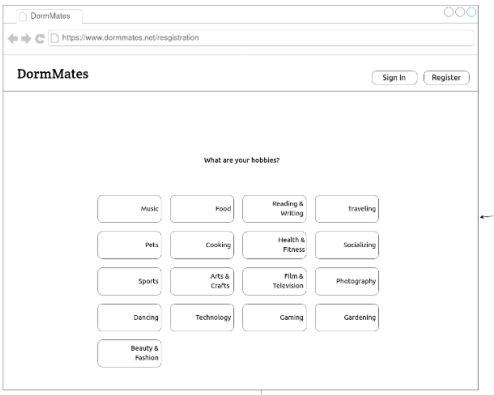
2. User selects the type of account they want to create

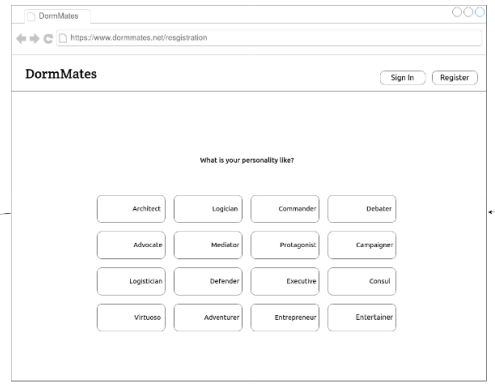


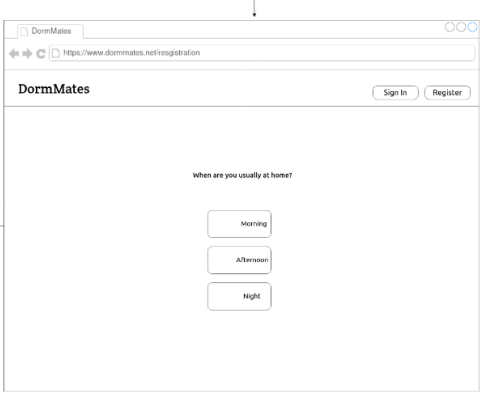
3. User creates a student account



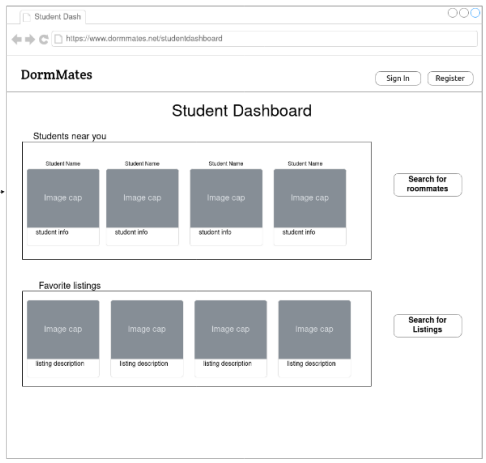
4. After creating a student account, student users must complete a questionnaire





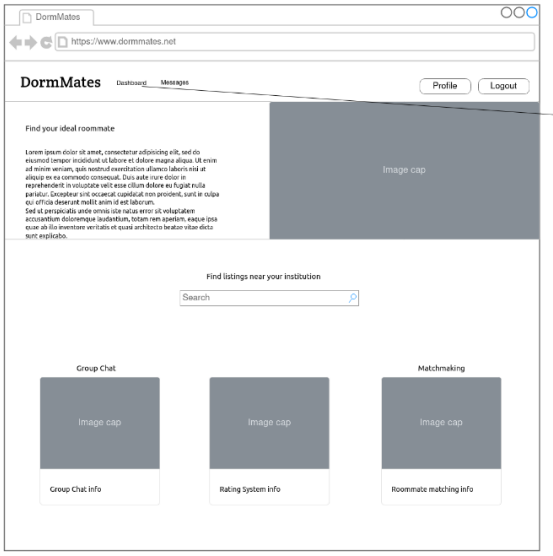


5. When student users complete their questionnaire, they get reccommendations for other students based on their questionnaire answers on the student dashboard

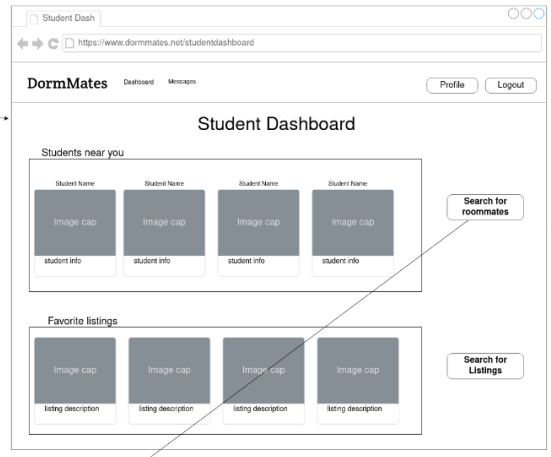


Use Case 5: Student Needs Roommates

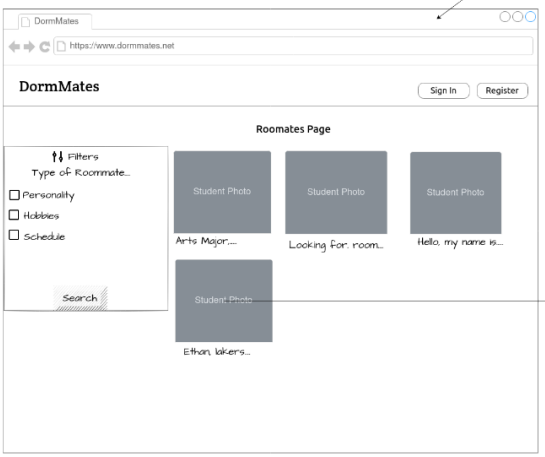
1. Student user is logged in on the homepage and clicks on dashboard in the nav bar



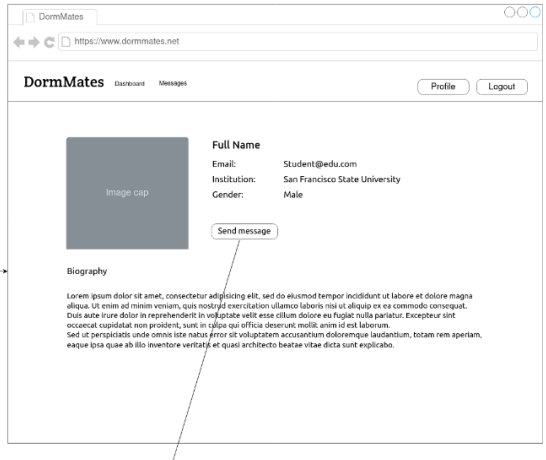
2. User is taken to the student dashboard and clicks on "search for roommates"



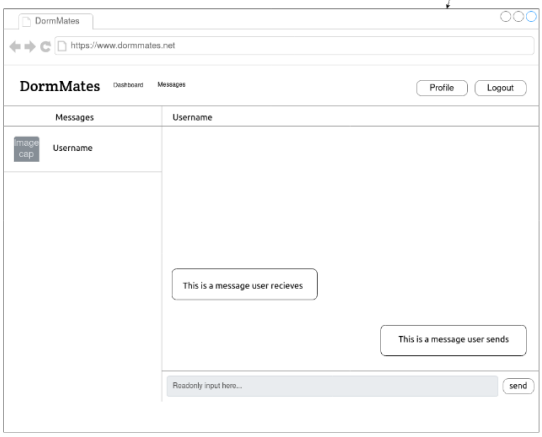
3. On the roommates page, users can click on individual user cards to go to specific users' profiles



4. On a specific user's profile, a user can choose to send them a message

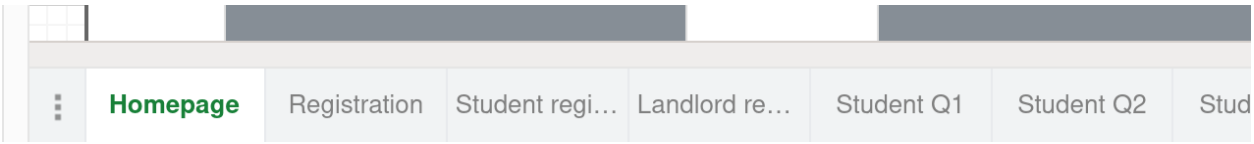


5. Users can message each other on the messages page



All Wireframes

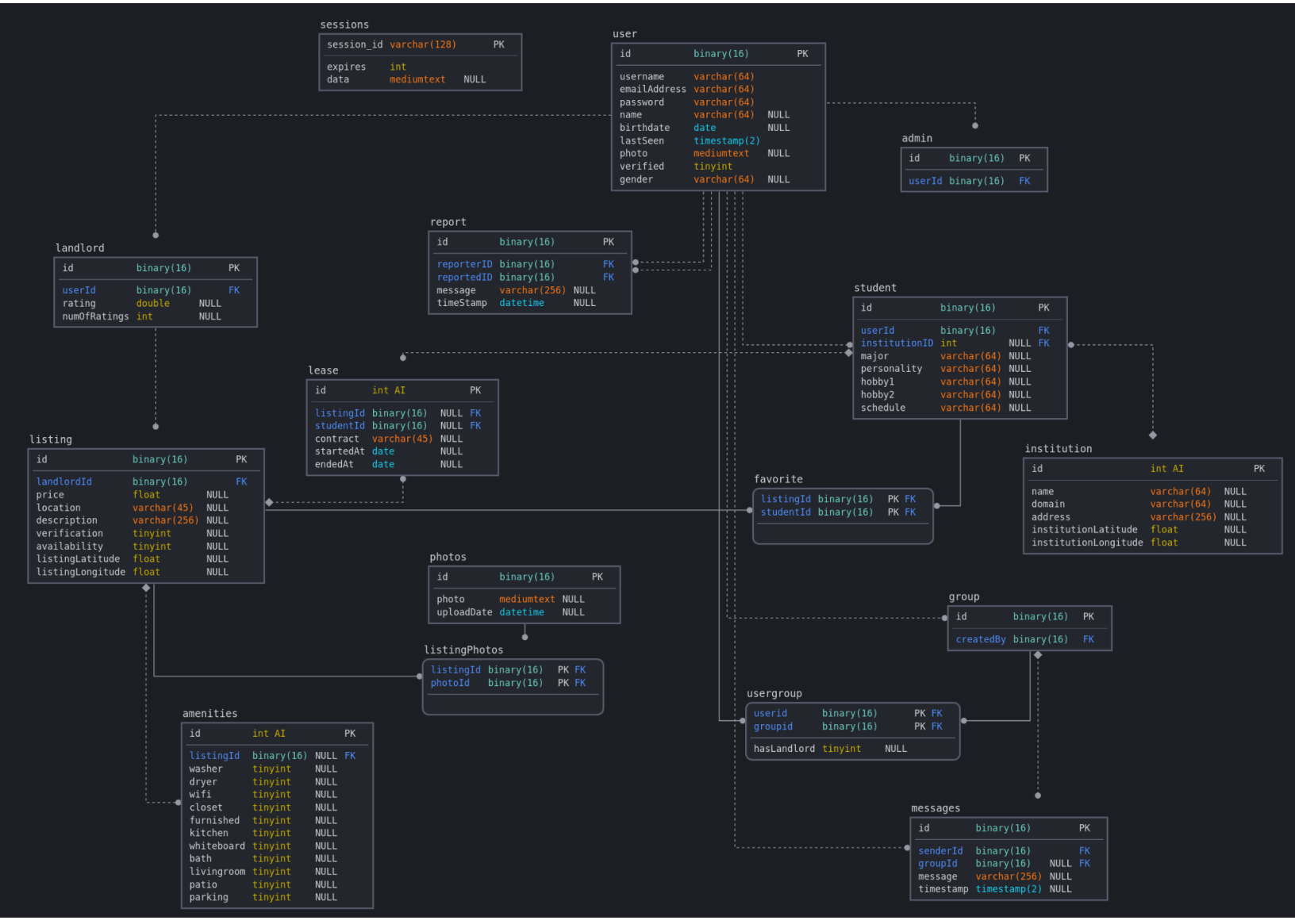
Each individual wireframe can be viewed here.
Each tab at the bottom of the web page represents one wireframe.



High Level Database Architecture and Organization V2

Database Model

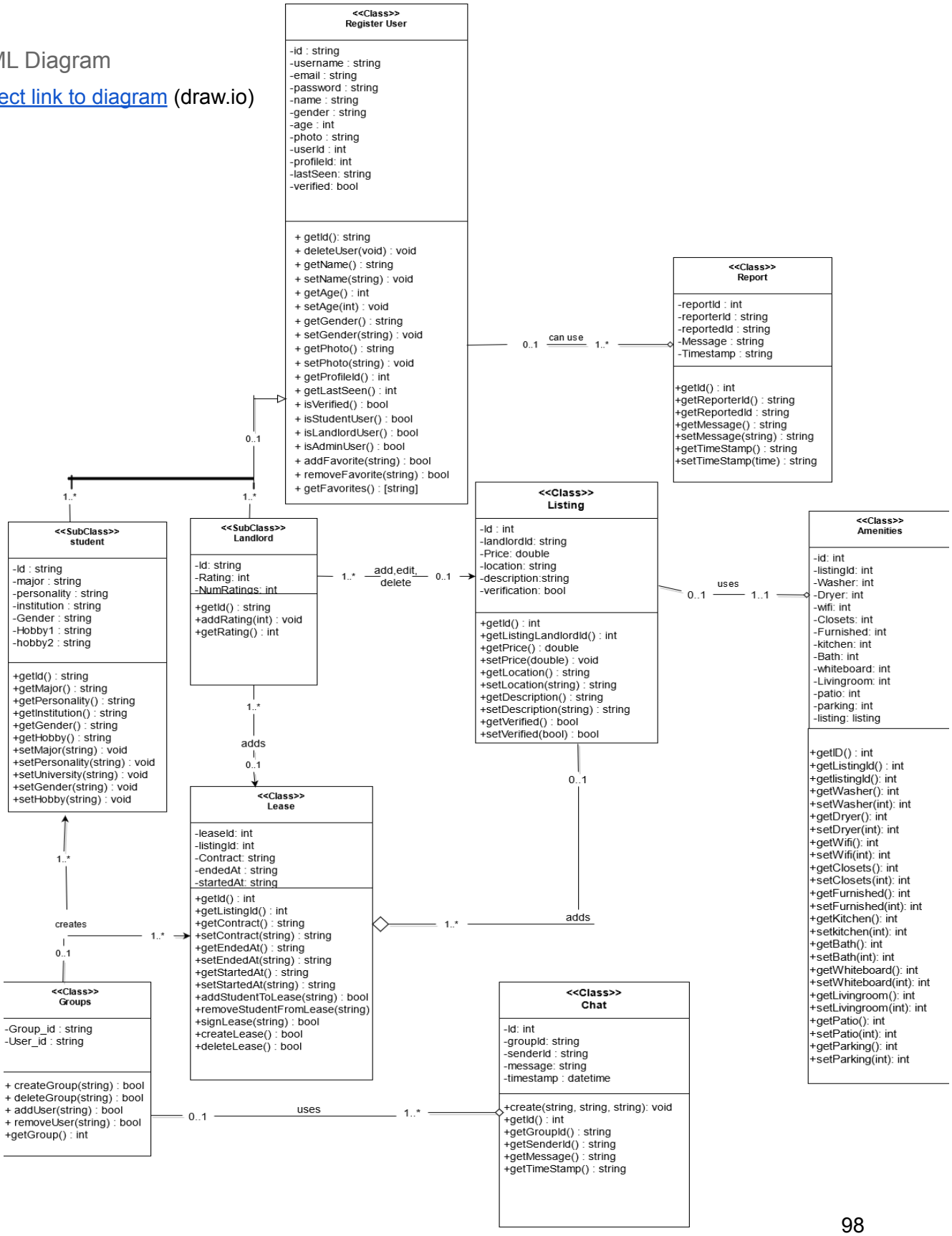
[DB SQL](pastebin.com) (pastebin.com)



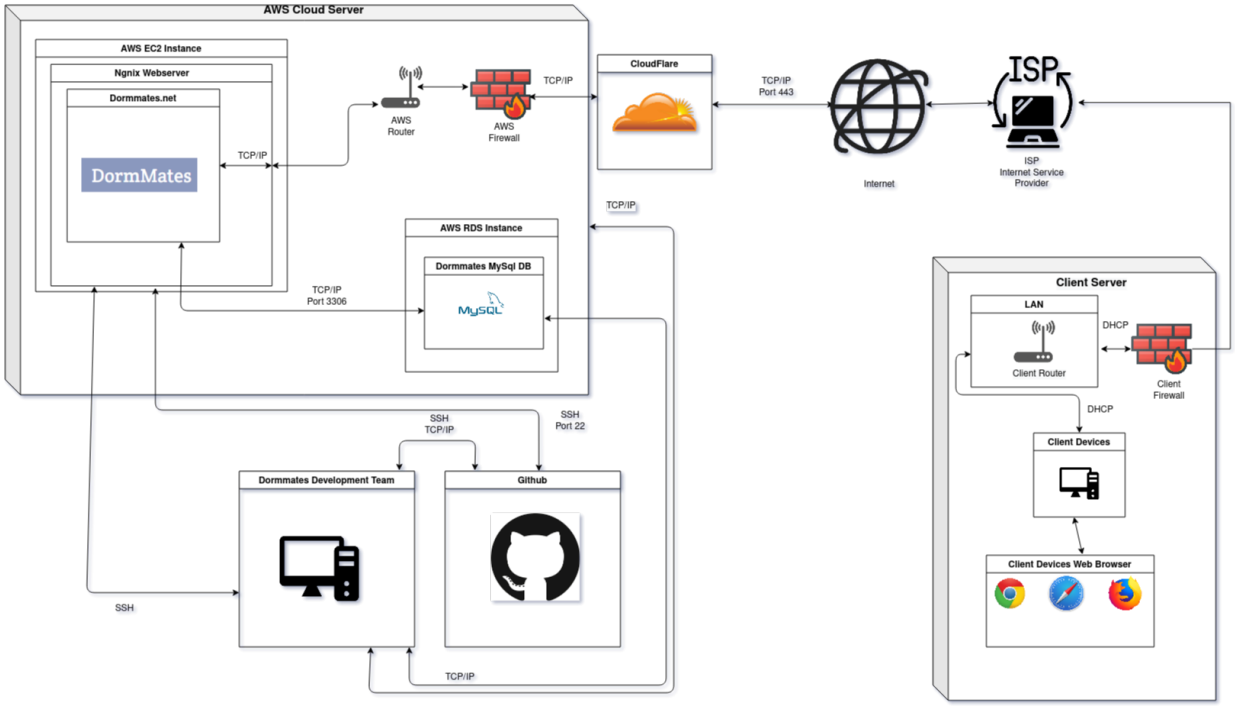
High Level Diagrams V2

UML Diagram

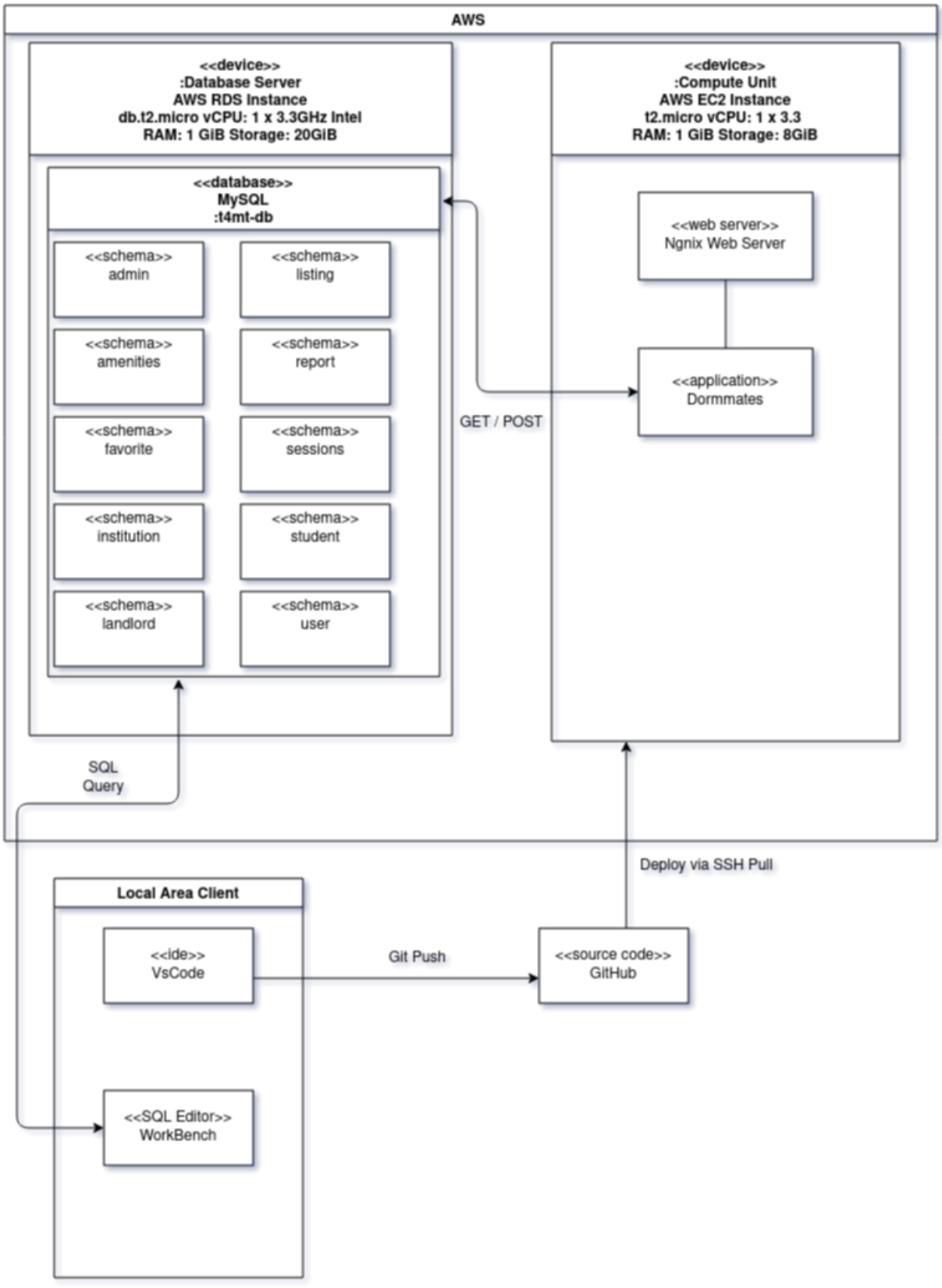
[Direct link to diagram](#) (draw.io)



Network Diagram



Deployment Diagram



Detailed List of Contributions

Andrei

- Hosted 1 team meeting where we discussed M2 feedback and worked on functional requirements V3.
- Created Jira tasks to address M2 feedback.
- Created Jira tasks for the frontend team to complete the horizontal prototype.
- Created Jira tasks for the backend team to work on numerous backend APIs.
- Hosted pair programming sessions with the backend team where we discussed implementation of key backend services.
- Hosted pair programming sessions with the frontend team where we discussed implementing wireframes.
- Gave feedback to the frontend and backend teams on their respective tasks.
- Performed code reviews on both the frontend and the backend.

Jonathan

- Participated in team meetings and worked on functional requirements V3.
- Worked on addressing business rules feedback.
- Hosted many pair programming sessions throughout the week with the backend team in order to implement key areas of the backend.
- Created backend documentation for the listing and leases API.
- Implemented the listing API and the lease API.
- Tested the listing API, lease API, search API.
- Assisted in testing and creation of search and favorites API.
- Updated and implemented Database models

Meeka

- Participated in team meetings and worked on functional requirements V3.
- Hosted pair programming sessions with the frontend team.
- Worked on addressing the vertical prototype feedback.
- Created wireframes for the profile, listing, and chat pages.
- Converted profile, listing, and chat wireframes into pug views.
- Created CSS used by the frontend team.
- Implemented scripts to make the frontend dynamic.
- Created dummy data that is used by the profile, listing, and chat pug views.

Jimmy

- Participated in team meetings and worked on functional requirements V3.
- Hosted pair programming sessions with the frontend team.
- Worked on addressing the UML diagram feedback.
- Created wireframes for the student and landlord dashboard pages.
- Converted student and landlord dashboard wireframes into pug views.
- Created dummy data that is used by the student and landlord dashboard pug views.

Alexandre

- Participated in team meetings and worked on functional requirements V3.
- Worked on addressing the Network and Deployment diagrams feedback.
- Hosted many pair programming sessions throughout the week with the backend team in order to implement key areas of the backend.
- Created backend documentation for the search and favorites API.
- Implemented the search and favorites API.
- Tested the search and favorites API.
- Assisted in creation of lease API and listing API.
- Assisted in testing lease API and listing API.

Sayed

- Participated in team meetings and worked on functional requirements V3.
- Worked on addressing the entities and ERD feedback.
- Created wireframes for the roommates, listings, and create-a-listing pages.
- Converted roommates, listings, and create-a-listing pages into pug views.
- Created dummy data for the roommates and listings pug views.

Ahad

- Participated in team meetings and worked on functional requirements V3.
- Created backend documentation for the chat and groups API.

Feedback Document

General notes for all the teams:

1. IMPORTANT: M3 is graded applying the following:
    a. Your grade in M3V2 document at the end of the semester (50%). If M3V1 has incomplete sections or poor work points will be taken off from M3 grade.
    b. If your vertical prototype (integrated) addressed all the feedback given by me then the points taken off will be recovered
    c. Horizontal Prototype (50%)
    d. Note, if you don't see feedback in one or more sections it is because the sections are good!.

M3 Docs Feedback

Section	Feedback
Main page and Table of Contents	
Data Definitions	
Prioritized Functional Requirements	They are good!. After M3, it is time to consider the scope of the features that are left to be implemented until the end of the semester, and start to think about dropping/adding P1s. I suggest being realistic with P1s, but at the same time if you need to drop P1s, make sure that it won't be affecting the quality of the product.
Wireframes	Creating an account & login wireframes are confusing, many connections crossing and It is difficult to determine the flow of how it works.
High Level Database Architecture and Organization	Every diagram must fit in one page. For example, your EER looks like it couldn't fit into the page. Providing links to the original

	diagram is not a bad idea, but the thing is that many teams will use printed copies of this to discuss some specific aspects of the database.
High Level UML Diagram	
High Level Application Network and Deployment Diagrams	
Team Feedback (sent by email from everyone in this team) A BIG thank you to everyone for putting in the effort to address all the problems found during my review of milestone 2. I have seen really good work in this milestone. I also would like to clarify that I already spoke with Ahad regarding his lack of contribution to this milestone.	Feedback: ● Pair programming sessions were extremely useful on both the frontend and backend teams. ● While most team members communicate almost daily through Discord, some members could be more active. ● Reading the documentation for any new library you are using is the best way to get familiar with it. Most, if not all libraries have great documentation and example code. Complaints: ● Ahad has been absent from the team for most of this milestone. He did not contribute much to helping the team or completing his assigned Jira tasks during this milestone. Here is my list of feedback: ● Jimmy was able to adjust quickly after being assigned to the frontend for this milestone and did a good job completing his tasks. ● Andrei was very helpful towards both teams and gave supportive feedback. He also assisted team members who got stuck on their part of the code and put in a lot of time to help solve the problem. It's impressive how he can keep up with both frontend and backend at the same time. ● Both teams made good progress to this

	milestone despite the short amount of time with the frontend working on the horizontal prototype and the backend working on the needed APIs. Feedback: The team did great this time around, lots of hours in discord working together. M3 was not too difficult but getting more code written and meeting functional requirements proved challenging, but the team did really well. Andrei / Front End team did really well, and overall, the people that were contributing really worked hard. Unfortunately since not everyone was contributing, some of the design was This milestone Ahad was not participating as much and had very little contributions to the backend. The amount of work and participation by Ahad seems to be very minimal. To be honest I'm not really sure as to what he has done. I feel as though every other member has spent extensive hours communicating with one another to accomplish the work this task had for us, but in contrast I saw little communication from Ahad and little participation. I'm thankful for the effort the rest of the team spent helping one another and time spent in individualized meetings to try and meet the requirements for the milestone. Overall, I think our team is doing good work and progressing well. Ahad has committed his chat api that he was assigned to finish at the last minute, and whether it works or not is left to be seen. But his lack of communication led to me being skeptical of his contributions as I never saw him join other members in their discord

	meetings outside of the mandatory team meetings as everyone has been participating in. Nor has he communicated in any of the messaging channels about the project. From what I could see with this information was someone that had done little while the rest of the team was working hard with one another. The front-end team is doing well. Everyone is working every day, so I don't see anything I can complain about. Team members like to overwork and get tired. Team members' tasks need to be assigned early and Better time management is needed. I suggest team members work on their task ASAP, to avoid last minute review and submission. Team is doing great with communication
Vertical Prototype:	• **I couldn't give you back the points lost in the vertical prototype because when I went to review the home page again the site was not loading.**
Horizontal Prototype:	**50/50** **Minor usability problems**

Feedback Given During Meeting

Horizontal Prototype Notes

- Fix for safari
- Buttons are contracting in Safari (remove type from buttons)
- Make description in the homepage bigger text, emphasize
- Format of the date is confusing for the user
- The spacing in between in the dashboard's students near me and favorite listings buttons

SW Engineering CSC648 Summer 2021

DormMates

Milestone 4 • Version 1 • 30 July 2021

Team 01

Team Lead and Github master	Andrei Georgescu	ageorgescu@mail.sfsu.edu
Frontend Lead	Meeka Cayabyab	
Backend & Database Lead	Jonathan McGrath	
Engineer	Alexandre Ruffo	
Engineer	Jimmy Yeung	
Engineer	Sayed Hamid	
Engineer	Ahad Zafar	

History Table

Version	Date	Notes
M4V2	08/03/2021	Addressed feedback and updated non-functional requirements status
M4V1	07/30/2021	Initial submission
M3V2	07/30/2021	Addressed feedback
M3V1	07/22/2021	Initial submission
M2V2	07/19/2021	Addressed the vertical prototype, functional requirements, and diagrams feedback.
M2V1	07/08/2021	Initial submission
M1V2	07/01/2021	Added database master role to the title page, addressed use cases feedback, addressed functional requirements feedback, and addressed competitive analysis feedback.
M1V1	06/22/2021	Initial submission

Table of Contents

Table of Contents **3**

Product Summary **4**

Final Priority 1 Functions 4

Unique Features 5

URL 5

Usability Test Plan **6**

QA Test Plan **13**

Code Reviews **16**

Self-check: Best Practices for Security **24**

Major Assets 24

Confirmation that PW in the DB are encrypted 24

Confirming Input Data Validation on the Backend 26

We also created helper functions that verify the uniqueness of a Username, Email. This is done by comparing the input with the database to ensure no duplicate emails or usernames. 27

Confirming Input Data Validation on the Frontend 28

Self-check: Adherence to Original Non-functional Specs **29**

Functionality 29

Security 29

Privacy 30

Legal 30

Performance 30

System Requirements 31

Marketing 31

Content 31

Scalability 32

Capability 32

Look and Feel 32

Coding Standards 33

Availability 34

Cost 34

Storage 34

Expected Load 35

Detailed List of Contributions **36**

Product Summary

DormMates

Final Priority 1 Functions

1.1. Unregistered User
- 1.1.1. An unregistered user can create a new student or landlord account.
- 1.1.2. An unregistered user can view listings near an institution of their choice.
- 1.1.3. An unregistered user can view the Home page.
- 1.1.4. An unregistered user can view the Terms of Service page.
- 1.1.5. An unregistered user can view the FAQ page.
- 1.1.6. An unregistered user can view the Features pages.
- 1.1.7. An unregistered user can view the About page.

1.2. Registered User
- 1.2.1. A registered user should be able to login to with their username and password.
- 1.2.2. A registered user should be able to logout of their account.
- 1.2.3. A registered user should be able to change their username.
- 1.2.4. A registered user should be able to change their email address.
- 1.2.5. A registered user should be able to change their password.

1.3. Students
- 1.3.1. A student user must have a verified edu email.
- 1.3.2. A student user must have a completed profile.
- 1.3.3. A student user shall be able to view the student dashboard.
- 1.3.4. A student user shall be able to view student user profiles.
- 1.3.5. A student user shall be able to search for listings.
- 1.3.6. A student user shall be able to edit their personality.
- 1.3.7. A student user should be able to edit their schedule.
- 1.3.8. A student user should be able to edit their hobbies.
- 1.3.9. A student user shall be able to filter listings by amenities.
- 1.3.10. A student user shall be able to filter listings by distance from university.
- 1.3.11. A student user shall be able to filter roommate selections by personality.
- 1.3.12. A student user shall be able to filter roommate selections by major.
- 1.3.13. A student user shall be able to filter roommate selections by hobbies.
- 1.3.14. A student user shall be able to filter roommate selections by schedule.
- 1.3.15. A student user should be able to view the location of a listing on a map.
- 1.3.16. A student user shall be able to favorite a listing.

1.4. Landlords

1.4.1. A landlord user shall be able to view the landlord dashboard.
1.4.2. A landlord user shall be able to view their own profile.
1.4.3. A landlord user shall be able to post listings.
1.4.4. A landlord user shall be able to edit their listings.
1.4.5. A landlord user shall be able to view student profiles.
1.4.6. A landlord user shall be able to delete their listings.

Unique Features

DormMates targets college students who would like to search for both housing and roommates. Students who sign up on our website will have the ability to look for potential roommates who share the same interests using our roommate filtering system. Students will additionally have the option to filter through housing listings to find their ideal housing. Those who want to list housing can create a landlord account and post a listing as well.

URL

https://dormmates.net

Usability Test Plan

Test Objective: Create a Listing

This test is to create a listing. The test asks users to add fields such as amenities, description, and a price to a listing that will be stored in the database. This is being tested as it is a core feature of the platform that landlord users should be able to perform.

Test Description:

The system setup requires that the user is already a landlord user and they will begin on their dashboard page. Users should be on the URL https://www.dormmates.net/dashboard to begin testing and will measure if the process is simple and easy to perform.

Usability Task Description:

TASK	DESCRIPTION
Task	Creating a Listing
Machine State	Listing is not created
Successful Completion Criteria	Listing is created and has the correct entered criteria
Benchmark	Completed in 1 minute

Test/Use Case	% Completed	Errors	Comments	Average Time Spent
Add a description to a listing	100%		There is no message saying what is the word limit for my description.	30 seconds average Within expected time window
Adding amenities to a listing	100%		Limited amount of amenities. Doesn't have an option to choose an amenity I have, but is not in the list.	20 seconds average Within expected time window
Adding a photo to a listing	60%	Photo is not displaying on the listing once it's created	Photo does not display once listing is created Displays a default picture instead of my photo	20 second average time Within expected time window

Test Objectives: Edit a Listing

This test is to allow a landlord to edit an already created listing. The test will ask users to update fields that they wish to change such as the amenities, price, or description. This is being tested because it's a major functionality that a landlord user should be able to perform.

Test Description:

The system startup requires that the user already has a landlord account and a listing created. A user will begin testing on the URL https://www.dormmates.net/dashboard and they will measure for how simple and easy it was to update and if it properly updated the listing.

TASK	**DESCRIPTION**
Task	Edit different traits of a listing
Machine State	A listing has already a specified amount of amenities
Successful Completion Criteria	Added and/or removed the proper amount of amenities that the user has updated.
Benchmark	Completed in 1 minute

Test/Use Case	**% Completed**	**Errors**	**Comments**	**Average Time Spent**
Edit Amenities of a listing	100%		Limited amount of amenities	2 min average Over expected time window
Edit price of a listing	100%		No specification for price	20 seconds average Within expected time window
Edit Description of a listing	100%		No specification for description limit Does not display listing description on listing card	30 second average time Within expected time window

Test Objective: Listing Favorites

This test is for the Favoriting function and its different uses. This is being tested because it's functionality for the student and also its purpose in displaying information about listings that students have favorited.

Test Description:

The system set up requires that the user is already a Student User. It also requires that the student be on a listing page or in order to view the favorites, on a student dashboard. The user should be on the URL https://www.dormmates.net/dashboard in order to start the testing. Intended users for the test are Student users who wish to favorite a listing for later viewing and for students who are on the dashboard and want to view their favorite listings on the student dashboard.

Usability Task Description:

TASK	**DESCRIPTION**
Task	Favoriting a Listing
Machine State	The listing is not favorited
Successful completion criteria	The listing has been favorited
Benchmark	Completed in a minute

Test/Use Case	**% Completed**	**Errors**	**Comments**	**Average Time Spent**
Favorite a Listing	100%		No message is displayed when I favorite a listing	1 min average Within time expected
View all created Favorites	100%		No message is displayed if I have no favorites	20 seconds average Within expected time window
Remove a Favorite Listing	90%		No message is displayed if I unfavorite a listing	2 min average Over expected time window

Test Objective: Listing API

This test is for the functionality of finding listings. We are testing the functionality of the filtering and all of it's resulting data also by query. This is tested due to it's imperative functionality for the student user and their ability to find nearby listings.

Test Description:

The system set up for this test requires the user to be a Student User and logged into the system. Also the user must be on the student dashboard. The starting point for the test is on the Student Dashboard. On the dashboard the user will navigate to the Search Listings page, and begin the test on the system.

Intended users for this test are users who are Student Users on the system and who are attempting to search for listings with different filters in their area. The user will begin the testing on the URL https://www.dormmates.net/dashboard.

Test Description:

TASK	**DESCRIPTION**
Task	Find a Listing
Machine State	Dashboard, No listings displayed
Successful completion criteria	Displays all listings that have a washer
Benchmark	Completed in 1 minute

Test/Use Case	**% Completed**	**Errors**	**Comments**	**Average Time Spent**
Find listing with wifi	100%	Map displays all listings still	Not that noticeable when search was completed	2 min average Over expected time window
Find listing with dryer and washer	100%	Map displays all listings	Not that noticeable when search was completed	20 seconds average Within expected time window
Find listings that are furnished	90%	Map displays all listings	Not that noticeable when search was completed	20 seconds average Within expected time window

Test Objective: Student API

This test is for the functionality of finding roommates. We are testing the functionality of the filtering and all of it's resulting data by query. This is tested due to it's imperative functionality for the student user.

Test Description:

The system set up for this test requires the user to be a Student User and logged into the system. Also the user must be on the student dashboard.The starting point for the test is on the Student Dashboard. On the dashboard the user will navigate to the Search Roommates page, and begin the test on the system.

Intended users for this test are users who are Student Users on the system and who are attempting to find other students in their area using specified filters. The user will begin the testing on the URL https://www.dormmates.net/dashboard.

TASK	DESCRIPTION
Task	Find Roommates
Machine State	No filters are being applied
Successful Completion Criteria	Displays all students with the proper personality
Benchmark	Completed in 1 minute

Test/Use Case	% Completed	Errors	Comments	Average Time Spent
Apply Major filters on students	100%		Only able to use one major filter. For example can't find students that are computer science majors or students that are physics majors	2 min average Over expected time window
Apply Personality filter on Student	100%		Only able to find with one personality filter Filters on personality not defined clearly	20 seconds average Within expected time window
Apply Schedule Filter on students	100%		Only able to use one schedule filter to find students Filters not clearly defined	20 second average

Questionnaire

Average of results of all testers

Questions	Strongly Disagree **1**	Disagree **2**	Neutral **3**	Agree **4**	Strongly Agree **5**
It was easy to edit a listing				X	
I found the systems too complex		X			
I found updating a description on a listing was easy to do					X
The fields were clear and concise with information needed to be entered					X
I was prompted with errors if I filled the fields in incorrectly				X	
Submitting each form was clear and easy				X	
It was easy and clear to favorite a listing				X	
It was easy and clear to view my favorites.					X
It was easy and clear to delete a favorite.					
Beginning each process was clear and intuitive					X
I found each process too complex.	X				

The filter options accurately filtered results					X
Results were easy to locate				X	
The results clearly show my filter options					X
The overall usability of this interface was excellent				X	

QA Test Plan

Test Objective

We will be testing the accuracy and stability of data to confirm that the site will be operating smoothly. Our unique features are based upon whether a user has registered an account as a student or a landlord therefore should be a prime focus on ensuring the quality of this process.

HW and SW setup

In order to access our website, users must have a working computer (Windows/MacOS) or a mobile device (iOS/Android) that has internet access. Users are able to access the website through browsers which includes Chrome, Safari, Microsoft Edge, and Mozilla Firefox. If a user signs up as a student account, the user must have a proper Edu email given to them by their institution. Once the credentials are met, the site can be reached by inputting https://dormmates.net/ in an eligible browser of their choice.

Feature to be Tested

1. The landlord dashboard page must only be available to landlord users.
2. The student dashboard page must only be available to verified student users.
3. All sensitive information must be encrypted before stored in the database.
4. Store users profile data on the database.
5. Store landlords listings on the database.

#	Description	Test Input	Expected Output	Pass/Fail
1	Landlord dashboard must only be available to landlord users.	Logged in student users will click on dashboard.	Lead the student user to the student dashboard page.	Pass
2	Student dashboard must only be available to student users.	Logged in landlord users will click on dashboard.	Lead the landlord user to the landlord dashboard page.	Pass
3	Unregistered users cannot access the dashboard.	dormmates.net/dashboard	Dashboard will not be shown in the navigation bar.	Pass
4	Attempting to create a Student user without an edu email.	User inputs a personal email, TestStudent@gmail.com	“University is not supported”	Pass
5	User creates password to be stored into the database as a hash	Password : “Jwalters34”	“Successfully created account” also a Hashed password in the database	Pass
6	User creates an account and types in a password.	Password : “Jwalters34”	Password will be hidden as it is typed in on the register page	Pass
7	Stores users ID as 36bit	Finish Registration Form	“Successfully created account”	Pass

	UUID	with correct fields	User Id will be encrypted hex in db	
8	Store users profile data on the database.	INSERT INTO user(id,username,emailAddress,password,name,birthdate,gender, photo, lastSeen) VALUES(UUID_TO_BIN(?, true),"Tommy","Tommy10@gmail.com",PassW0rd21,"Tom Holland",9-10-1990,Male, Photo,NOW())	"Successfully created account"	Pass
9	Store Students profile data on the database	INSERT INTO student(id,userId,institutionID,major,personality,schedule,hobby1,hobby2) VALUES(UUID_TO_BIN(*, true),UUID_TO_BIN(*, true),economics,mediator, afternoon,dancing,gardening)	"Student created" Along with a student object with the parameters passed	Pass
10	Store landlords profile data on the database	INSERT INTO landlord(id,userId,rating,numOfRatings) VALUES(UUID_TO_BIN(*, true),UUID_TO_BIN(*, true), 0, 0)	"Landlord created" Along with a landlord object	Pass
11	Store Landlords listing data on the database	INSERT INTO listing(id,landlordId,price,location,description,verification,availability,listingLatitude,listingLongitude) VALUES(UUID_TO_BIN(*, true),UUID_TO_BIN(*, true),1000,description,true,true,33,-121)	"Listing Created" Along with a listings object with the parameters passed	Pass

12	Store Listings Amenities on the database	INSERT INTO amenities(listingId,washer, dryer,wifi,closet,furnished, kitchen,whiteboard,bath,liv ingroom,patio,parking) VALUES(UUID_TO_BIN(*, true),0,1,1,1,1,0,1,1,1,true, false)	“Amenities Created” and “Listing Created” Along with the parameters that were passed	Pass
13	Remove Listing from database from landlord	DELETE FROM listing WHERE id = UUID_TO_BIN(*,true)	“Listing deleted”	Fail
14	Delete listings Amenities from database	DELETE FROM listing WHERE id = UUID_TO_BIN(*,true)	“Listing deleted”	Fail
15	Viewing favorite listings	Click on student Dashboard	Listings displayed	Pass

Code Reviews

Coding Style:

- File names will be universal with their naming conventions and entities
- Code will be organized and labeled appropriately with comments
- Headers will be clear and include file names and descriptions

BackEnd:

- Backend code will use camel case naming convention
- Helper functions will be notated and implemented where needed
- Naming convention will be universal through every route to model
- Parameters will match the naming convention of the database entities and attributes

Front End:

- Button tags will have matching names that follows the backend
- Div id will be unique so that they are to be used in javascript accordingly

AR

Alexandre Ruffo
Fri 7/30/2021 2:12 PM
To: Jonathan Michael Mcgrath

Hello Jonathan,

To answer your question on why it is necessary to delete keys when the type of value is undefined is because of how we filter our search. When a value in the amenities is found to be undefined that means a user has not selected them as a search filter option. We delete the key from the map and continue to check if any of the other keys have a value that is undefined. We then return all matching listing that have the matching filter options selected by the user.

To my knowledge there was no other way that we discussed.

Thank you,
Alexandre Ruffo

...

Reply | Forward

JM

Jonathan Michael Mcgrath
Fri 7/30/2021 1:58 PM
To: Alexandre Ruffo

Good Afternoon,

I checked out your code and everything looks good, it fits the coding style we discussed. Here are my questions and feedback

Feedback:
Nice work on the filtering using concatenation, this looks very clean and concise.

Good Job on validation also.

I do think this code could have more comments, but over all it's done very well.

Questions:
Can you explain why we delete each key when it is undefined?
Was there a different way we planned on implementing this?

Thank You,
Jonathan McGrath

...

AR

Alexandre Ruffo
Fri 7/30/2021 12:49 PM
To: Jonathan Michael Mcgrath

Hello Jonathan,

Below I have provided all functions that pertain to the roommate search

Roommate Search Model

```
/*********************************************************************
 * This function queries the database to find all rows in the student
 * that match the given parameters chosen by the user.
 * Returns: an array of students
 *********************************************************************/

const getRoommatesByFiltering = async function (filters) {
  if (!(Object.values(filters).length === 0)) {
    const sqlPrepend = 'SELECT BIN_TO_UUID(student.id,true) AS studentId, BIN_TO_UUID(user.id,true) AS userId, institution.name AS institution, student.major, student.personality
    const sqlJoin = ' INNER JOIN institution ON institution.id = student.institutionID INNER JOIN user ON user.id = student.userId WHERE '
    const sqlFilters = Object.keys(filters).map(k => `student.${k} = ?`).join(' AND ')
    // Query the database and return students
    const [students, fields] = await db.query(`${sqlPrepend}${sqlJoin}${sqlFilters}`, Object.values(filters));
    return students;
  }

  const sql = `
    SELECT
      BIN_TO_UUID(student.id,true) AS studentId,
      BIN_TO_UUID(user.id, true) AS userId,
      institution.name AS institution,
      student.major,
      student.personality,
      student.hobby1,
      student.hobby2,
      student.schedule,
      user.username,
      user.emailAddress,
      user.name,
      user.birthdate,
      user.lastSeen,
      user.photo,
      user.verified,
      user.gender
    FROM student
    INNER JOIN user ON user.id = student.userId
    INNER JOIN institution on institution.id = student.institutionID
  `;
  const [students, fields] = await db.query(sql);
  return students
}
```

```
/**************************************************************************
 * This function is designed to be used to filter a search for
 * student users by using query parameters. It searches the database
 * any students that have the same parameters that match their attributes
 * Returns:
 * 200 if the request is successful
 * 400 if the request is unsuccessful
 *************************************************************************/


const getRoommateByFiltering = async function (req, res, next) {
  //map of roommate filter options with a validation check to see if the values are not null
  let filters = {
    major: req.query.major,
    personality: req.query.personality,
    hobby1: req.query.hobby1,
    hobby2: req.query.hobby2,
    schedule: req.query.schedule
  };

  //map of majors
  let majors = {
    computerScience: 'computer science',
    physic: 'physics',
    mathematic: 'mathematic',
    biology: 'biology',
    businessManagement: 'business management',
    business: 'business',
    accounting: 'accounting',
    nursing: 'nursing',
    psychology: 'psychology',
    communication: 'communication',
    marketing: 'marketing',
    generalEducation: 'general education',
    elementaryEducation: 'elementary education',
    finance: 'finance',
    criminalJustice: 'criminal justice',
```

```
  criminalJustice: 'criminal justice',
  politicalScience: 'political science',
  economics: 'economics',
  electricalEngineering: 'electrical engineering',
  history: 'history',
  liberalArts: 'liberal arts',
  sociology: 'sociology'
};

//map of personalities
let personalities = {
  architect: 'architect',
  logician: 'logician',
  commander: 'commander',
  debater: 'debater',
  advocate: 'advocate',
  mediator: 'mediator',
  protagonist: 'protagonist',
  campaigner: 'campaigner',
  logistician: 'logistician',
  defender: 'defender',
  executive: 'executive',
  consul: 'consul',
  virtuoso: 'virtuoso',
  adventurer: 'adventurer',
  entrepreneur: 'entrepreneur',
  entertainer: 'entertainer'
};

//map of hobbies
let hobbies = {
  music: 'music',
  food: 'food',
  readingWriting: 'reading/writing',
  travel: 'travel',
```

```
  pets: 'pets',
  cooking: 'cooking',
  healthFitness: 'health and fitness',
  socializing: 'socializing',
  sports: 'sports',
  artsCrafts: 'arts and crafts',
  filmTv: 'film and television',
  photography: 'photography',
  dancing: 'dancing',
  technology: 'technology',
  gaming: 'gaming',
  gardening: 'gardening',
  beauStyFashion: 'beauty and fashion',

};

//map of schedules
let schedules = {
  morning: 'morning',
  afternoon: 'afternoon',
  night: 'night'
};

//validation of all the map values
for (const [key, value] of Object.entries(filters)) {
  if (key === 'major') {
    if (typeof value === 'undefined') {
      delete filters[key];
    }
    else if (majors[value] !== 'undefined') {
      filters[key] = majors[value];
    } else {
```

```
  } else {
    return res.status(404).status({
      error: 'Invalid major provided'
    })
  }
}
else if (key === 'personality') {
  if (typeof value === 'undefined') {
    delete filters[key];
  }
  else if (personalities[value] !== 'undefined') {
    filters[key] = personalities[value];
  } else {
    return res.status(404).status({
      error: 'Invalid personality provided'
    });
  }
}
else if (key === 'schedule') {
  if (typeof value === 'undefined') {
    delete filters[key];
  }
  else if (schedules[value] !== 'undefined') {
    filters[key] = schedules[value];
  } else {
    return res.status(404).status({
      error: 'Invalid schedule provided'
    });
  }
}
else if (key === 'hobby1') {
  if (typeof value === 'undefined') {
    delete filters[key];
  }
  else if (hobbies[value] !== 'undefined') {
```

```
    }
    else if (hobbies[value] !== 'undefined') {
      filters[key] = hobbies[value];
    } else {
      return res.status(404).status({
        error: 'Invalid hobby provided'
      });
    }
  }
  else if (key === 'hobby2') {
    if (typeof value === 'undefined') {
      delete filters[key];
    }
    else if (hobbies[value] !== 'undefined') {
      filters[key] = hobbies[value];
    } else {
      return res.status(404).status({
        error: 'Invalid hobby provided'
      });
    }
  }
}

try {
  const students = await Search.getRoommatesByFiltering(filters)

  return res.status(200).send({
    students
  });

} catch (error) {
  console.log(error);
  return res.status(400).send({
    message: 'Error in the request'
  });
}
```

```
//Find all roommates by filters
router.get('/student', (req,res,next) =>{
  SearchController.getRoommateByFiltering(req,res,next);
});
```

Self-check: Best Practices for Security

Major Assets

- Sensitive User Data (emails & attending university)
- Passwords
- Listing IDs
- Landlord IDs
- Student IDs
- User IDs

Confirmation that PW in the DB are encrypted

First and foremost, selecting all user rows from our database confirms that passwords are encrypted.

id binary	username varchar	emailAddress varchar	password varchar
@Hgک ��ς��)�e	andreil	andrei@gmail.com	$2b$10$iFLHTwPUBDPwCrP1d8CeBePJigLT9uiUYKjHNZqDKHtsIauOCUX9.
C*$`B���LO��X	Jon10	jonny@sfsu.edu	$2b$10$/7FwqsvyBDMMy2UQGdTJh.ya/cjBDaMC4fnlfQmsyW4RemiGVXX1W
Ef#���=���]�~	andreis	as@sfsu.edu	$2b$10$FKclGUonlvCQMUuymyqqg.w9acrFeggF9lKHAbpdJtwOqMdTOwggm
FfG�k��z<<<	iLoveTea	alexispoop@gmail.com	$2b$10$f5FHV.06QZlbqeXmB9uV4upAQj8BFz9VFLdfziTiVDnyZrxwXSfSm
G&6 .}w	andrei	ageorgescu@sfsu.edu	$2b$10$/abDhC.rTmfENPKni9ZH8ez2oBmnvjSY/lHOD2yJm2pkW/wtekU2i
H'~PV?	qwe	ar@sfsu.edu	$2b$10$M6MBe9YWgLY0zG3FVrJ1QeYFCnoM/DsLTb.x6ysPYZCgTTftGew2S
H-7;K	jortizco	jortizco@sfsu.edu	$2b$10$0WIXbwg4WR5m6MSbGDQ9Ru/ujXPRaHpLWx85Ap/DgPU9tvxPKWKDe
OI0c8IE)	Jon11	jon11@gmail.com	$2b$10$MWRr27z57twOKRHKqGtZE.ccAAb8Bms1A/35g4Hh/t5Wmjt4DHr/i

As for the process; at a high-level it consists of the password being sent to the Auth API and it being hashed before our User model inserts it into the database.

This process all starts with the '/auth/register' route which is a POST route. This route simply calls the 'createUser' method from the User Controller.

```
router.post("/register",(req, res, next) => {

  UserController.createUser(req, res, next);

});
```

The `createUser` method first calls a method that validates the request's body and then if that succeeds it calls another method that interacts with the User model.

```
// ---------------------------------------------------------------
// This function takes in a request body and creates a new user.
// Returns:
//      200 if successful, 400 if something went wrong
//      JSON with a message field
// ---------------------------------------------------------------
const createUser = async function (req, res, next) {
  // Validating registration form
  const bodyValidated = await validateRegistrationBody(req, res, next);
  if(bodyValidated === true) {
    await registerNewUser(req, res, next);
  }
};
```

Inside of the `registerNewUser` method is where a user's password gets encrypted. Specifically we use **Bcrypt** which is a fairly secure password hashing algorithm.

```
  try {
    // Hashing the password and creating the user
    const hashedPassword = await hashPassword(password);
    const userCreated = await User.createNewUser(
      username,
      email,
      hashedPassword,
      name,
      dob,
      gender,
    );

    if (userCreated) {
      return res.status(200).send({
        error: null,
        message: 'User Created',
        id: userCreated,
      });
    }

    // If the user was not created, return an error
    return res.status(200).send({
      error: 'Could not create a new user.',
      message: 'User not created',
    });

  } catch (e) {
    console.log(e);
    return res.status(200).send({
      error: 'Could not create a new user.',
      message: 'User not created',
    });
  }
}
```

It's important to note here that we placed the `hashPassword` method inside of a try-catch block. This means that if anything goes wrong during the hashing process, the user's data will never get inserted into the database and therefore we avoid accidentally storing unhashed passwords.

Confirmation of User, Student, Landlord and Listing IDs

For the IDs of each entity, we used a function to generate a UUID (Universally Unique ID)to create a binary key. This key adds extra security by generating a 128-bit number to identify each entity.

application > backend > utils > JS uuid.js > <unknown> > exports

```
//Produces a UUID
//Credit to broofa from stackoverflow
//https://stackoverflow.com/questions/105034/how-to-create-a-guid-uuid/2117523#2117523
module.exports = function () {
    return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function (c) {
        var r = Math.random() * 16 | 0,
            v = c == 'x' ? r : (r & 0x3 | 0x8);
        return v.toString(16);
    });
}
```

Everytime we create one of these new entities, we use the UUID() function to generate this key and store it into the DB.

```
const createNewUser = async function(username,email,password,name,dob,gender){
    const id = uuid();
    const sql = `
        INSERT INTO
        user(id,username,emailAddress,password,name,birthdate,gender,lastSeen)
        VALUES(UUID_TO_BIN(?, true),?,?,?,?,?,?,NOW())
    `;
    const data = [id,username,email,password,name,dob,gender];
```

Confirming Input Data Validation on the Backend

In the backend we perform validation checks for when a user registers for an account on our service.

```
// Validate the user data
if (!username || !email || !password || !name || !dob || !gender || !type || !avatar) {
  return res.status(200).send({
    error: 'Form is incomplete',
    message: 'Please fill in all required fields',
  });
}
```

In the code above we check if all values within the registration form are fully completed and ask the user to fill in all fields of the form if they are not.

```
// Check if the email is a valid email
if (!email.match(/^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$/i)) {
  return res.status(200).send({
    error: 'Email address does not meet requirements',
    message: 'Please enter a valid email address',
  });
}
```

Within the code above we check whether the email they input is of a valid format and if it fails then we ask them to re-enter a valid format.

```
// Check if the password is at least 8 characters long and contains at least
// one uppercase letter and one number
if (!password.length >= 8 || !password.match(/([A-Z])/)) {
  return res.status(200).send({
    error: 'Password does not meet requirements',
    message:
      'Please enter a password with at least 8 characters, one uppercase letter and one number',
  });
}
```

We check if the user enters in a password that meets a format of a minimum of 8 characters lengths and 1 uppercase letter with 1 number in the password. If it fails we display a message stating as such and have the user enter in a valid password.

We also created helper functions that verify the uniqueness of a Username, Email. This is done by comparing the input with the database to ensure no duplicate emails or usernames.

```
    // Validating if the email is unique
    const emailIsUnique = await User.getByEmail(email);
    if (emailIsUnique) {
      return res.status(200).send({
        error: 'Could not create an account with that email.',
        message: 'Email is taken.',
      });
    }
  } catch (e) {
    return res.status(200).send({
      error: 'Could not create a new user.',
    });
  }

  return true;
}
```

Confirming Input Data Validation on the Frontend

In the frontend, the search bar input is created on the home pug view using form.

```
form#institution-search-form.searchbar(role='search')
    .form-group
        span.error(id='error')
        input.form-control(type='text' placeholder='Enter Institution Name' id='institution-search' autocomplete="off")
        ul.list-group(id='institution-search__searchField')
```

The search bar is used to search for a user's desired institution and was implemented in the following javascript function.

```
function search(event) {
  const resultsContainer = document.querySelector(
    '#institution-search__searchField',
  );

  if (!event.target.value.length) {
    resultsContainer.innerHTML = '';
    return;
  }

  fetch(`/search/institution/${event.target.value}`, {
    method: 'GET',
    headers: {
      'Content-Type': 'application/json',
    },
  })
    .then((response) => response.json())
    .then((data) => {
      // data is an array of institutions
      const { institutions } = data;

      // If there are no results, display an appropriate message
      if (institutions.length === 0) {
        resultsContainer.innerHTML = 'No institutions found.';
        return;
      }

      // Render the results
      const resultsList = institutions.map((element) => `
        <li class="list-group-item list-group-item-action" id="${element.id}" onclick="selectInstitution('${element.name}')">
          ${element.name}
        </li>
      `);
      resultsContainer.innerHTML = `${resultsList.join('')}`;
    })
    .catch(() => {
      resultsContainer.innerHTML = 'No institutions found.';
```

The backend has a list of institutions which is accessed in the frontend by using fetch. Once the user begins to fill the input, the code will then check the value's length to see if the data is available in the backend. Based on the user's inputs, it will display the data results and no results if none.

Self-check: Adherence to Original Non-functional Specs

Functionality

Requirement	Status
The website should utilize all tools and frameworks approved by the CTO.	DONE
The website should be easy to use and intuitive.	DONE
The website should have a simple and non-cluttered interface.	DONE
The website should be responsive across all modern devices.	DONE
The website will use Amazon Web Services for deployment.	DONE
The website will use Amazon Web Services for its database.	DONE
The website should use HTTPS for all requests.	DONE

Security

Requirement	Status
Users must authenticate themselves before accessing any protected pages.	DONE
Users must authenticate themselves if their cookie is expired.	DONE
The student dashboard page must only be available to verified student users.	DONE
The landlord dashboard page must only be available to landlord users.	DONE
Registered users should be able to view their own chat messages.	ISSUE*
Registered users should be able to send messages only within their group.	ISSUE*
All sensitive information must be encrypted before stored in the database.	DONE

* Did not have enough time to implement chat on the frontend. The backend code for chat was delivered late and therefore we did not have enough time to implement it fully.

Privacy

Requirement	Status
Only registered users will be able to view all listings.	DONE
Only registered users will be able to view students.	DONE
Landlords will not have access to viewing other landlords.	DONE
Landlords will not be able to search student data.	DONE
Landlords will not be able to search landlords.	DONE
Registered users' chat messages should remain private.	ISSUE*

* Did not have enough time to implement chat on the frontend. The backend code for chat was delivered late and therefore we did not have enough time to implement it fully.

Legal

Requirement	Status
All users must accept the terms and service policy before creating an account.	DONE
All users must accept the privacy policy before creating an account.	DONE
All landlords must prove ownership of a listing.	ISSUE*
The website must have a copyright notice.	DONE
The website must have a privacy policy notice.	DONE
The website must have a terms and conditions notice.	DONE
The website must have a cookie notice.	DONE
All content uploaded to the site must be owned by the user who is uploading it.	DONE

* Ran out of time implementing the admin dashboard which was responsible for "approving" or "denying" valid listings that were posted by landlords.

Performance

Requirement	Status
The frontend must have processes in place that prevent it from being offline.	DONE
The backend must have processes in place that prevent it from being offline.	DONE
The website load time should be within industry standard requirements.	DONE

System Requirements

Requirement	Status
The website shall work up to version 91.0.4472.106 of Google Chrome.	DONE
The website shall work up to version 14.0.03 of Safari.	DONE
The website shall work up to version 91.0.864.48 of Microsoft Edge.	DONE
The website shall work up to version 85.0 of Mozilla Firefox.	DONE
The website shall work up to version 11.0 of Android.	DONE
The website shall work up to version 14.6 of iOS.	DONE
The website will be supported in the English language.	DONE

Marketing

Requirement	Status
The website should follow SEO best practices.	DONE
Each page on the website shall have the logo on the navigation bar.	DONE
Each page will be clear and easy to navigate for new visitors.	DONE
Each user shall be able to connect their account with their social media platforms.	ISSUE*

* Did not have enough time to implement this because we realized it was missing towards the end of the project and would have required a lot of refactoring. We chose to focus on implementing key features instead.

Content

Requirement	Status
The website will have a navigation bar.	DONE
The website navigation bar will direct users to different pages.	DONE
The website pages will have a footer.	DONE
The website will have a scalable map.	DONE
The website should give registered users the option to private message.	ISSUE*

* Did not have enough time to implement chat on the frontend. The backend code for chat was delivered late and therefore we did not have enough time to implement it fully.

Scalability

Requirement	Status
The website should be capable of handling a large number of listings.	DONE
The website should be composed of a frontend and backend which are separate codebases.	DONE
The website shall be able to handle a large number of users.	DONE
The chat rooms shall be able to handle a large number of users.	DONE

Capability

Requirement	Status
The website should process all requests as expected by the users.	DONE
The website should respond with a descriptive error if one occurs.	DONE
The website should alert users when they are about to leave the site.	DONE

Look and Feel

Requirement	Status
The navigation bar should have a logo.	DONE
The navigation bar should have a dark colored background.	DONE
The navigation bar should have a light shade hover color button.	DONE
The footer should have a logo.	DONE

The footer should have a sitemap with all site pages.	DONE
The website should have a plain color layout.	DONE
The website should have a simple layout.	DONE
The website will have a readable font.	DONE
The website elements fonts will be uniform.	DONE
The website elements will be continuous.	DONE
The website’s pages will be scrollable in the vertical axis.	DONE
The font should be roman new times.	DONE
The feeling should be friendly.	DONE
The website should not be repetitive.	DONE
The website should be easy to traverse.	DONE
Pages should be instant loaded.	DONE
The private account page should be easy to find.	DONE
The private chat font will be easy to read and uniform.	DONE
The private chat should be easily identifiable.	DONE
The map should be easily identifiable.	DONE
The map key will be easily identifiable.	DONE
Profiles will clearly display a user’s role.	DONE
The post should be easily identifiable.	DONE
The forum should be easily identifiable.	DONE
The buttons should be easily identifiable.	DONE
Listing filters should be easily identifiable.	DONE

Coding Standards

Requirement	**Status**
All code must be reviewed before it is merged with any of the three main branches.	DONE
All code must be submitted via pull requests.	DONE
All code must be pushed to proper branches.	DONE

All code must be documented.	DONE
All code should be organized.	DONE
There should be no repetitive code.	DONE
There should be no unused code.	DONE
All code should have in-line comments where needed.	DONE
The code should have a uniform formatting style.	DONE
The backend code should use an object-oriented programming paradigm.	DONE
The backend must implement methods to prevent SQL injection.	DONE

Availability

Requirement	**Status**
The frontend must be online at all times.	DONE
The backend must be online at all times.	DONE
The website is updated if and only if code is pushed to the master branch.	DONE
The website will resync if a loss of connection occurs.	DONE
The website shall display error messages when errors occur.	DONE
The website will be managed on a PST timezone.	DONE

Cost

Requirement	**Status**
Amazon web services server is free.	DONE
Amazon web services relational database is free.	DONE
Server must not exceed the free tier.	DONE
Server maintenance is free.	DONE

Storage

Requirement	Status
Store users profile data on the database.	DONE
Store landlords listings on the database.	DONE
Remove listings from the database after it has been deleted by the user.	DONE
Store up to 60 days of inactive listings (incase user wants to repost).	DONE
Remove listings from the database after 60 days of inactivity.	ISSUE*
Repost will restart the 60-day clock of storage time.	ISSUE*
Store students' chat history on the database.	ISSUE*
Store usernames on the database	DONE
Store emails on the database	DONE
Store passwords on the database	DONE
Store landlord information on the database.	DONE
Store landlord photos on the database.	DONE
Store student photos on the database.	DONE
Store error logs on the database.	ISSUE*

* Did not have enough time to implement chat on the frontend. The backend code for chat was delivered late and therefore we did not have enough time to implement it fully.
* We decided that storing logs on the machine itself was better and easier to implement.

Expected Load

Requirement	Status
The website will be able to handle as many users as AWS can support.	DONE
The website will be able to handle as many listings as AWS can support.	DONE

Detailed List of Contributions

Andrei

- Hosted 1x team meeting where we worked on the product summary, discussed priority 1 functional requirements, and other milestone 4 related tasks.
- Hosted pair programming sessions with the frontend and the backend teams.
- Edited and formatted the milestone 4 document.
- Worked on connecting the frontend to the backend api.

Jonathan

- Hosted meeting with the backend team to discuss Usability test plan.
- Hosted multiple pair coding sessions to get back end code working.
- Worked on the Usability test plan.
- Worked on the QA Test Plan.
- Worked with Alexandre on Code review.
- Worked on the P1 compromise
- Worked on Self Check Best Practices for Security

Meeka

- Hosted meeting with the frontend team to discuss frontend feedback from the entire team.
- Hosted meeting with the frontend team to discuss QA test plan.
- Worked on the QA Test Plan
- Worked on Self Check Best Practices for Security
- Worked on Unique features for Product Summary
- Implemented feedback given by the team on the FAQ, listing, profile, and search pages.

Jimmy

- Implemented feedback given by the team on the dashboard page.
- Implemented Roommates Near you
- Participated in pair programming
- Added loading animations to frontend
- Added fake data to the production site
- Created datepicker for the registration page
- Fixed various bugs on the frontend

Alexandre

- Worked on the Usability test plan.
- Worked on Code review with Jonathan.
- Participated in coding sessions to get back end code working.
- Worked on Self Check Best Practices for Security
- Worked on the P1 compromise

Sayed

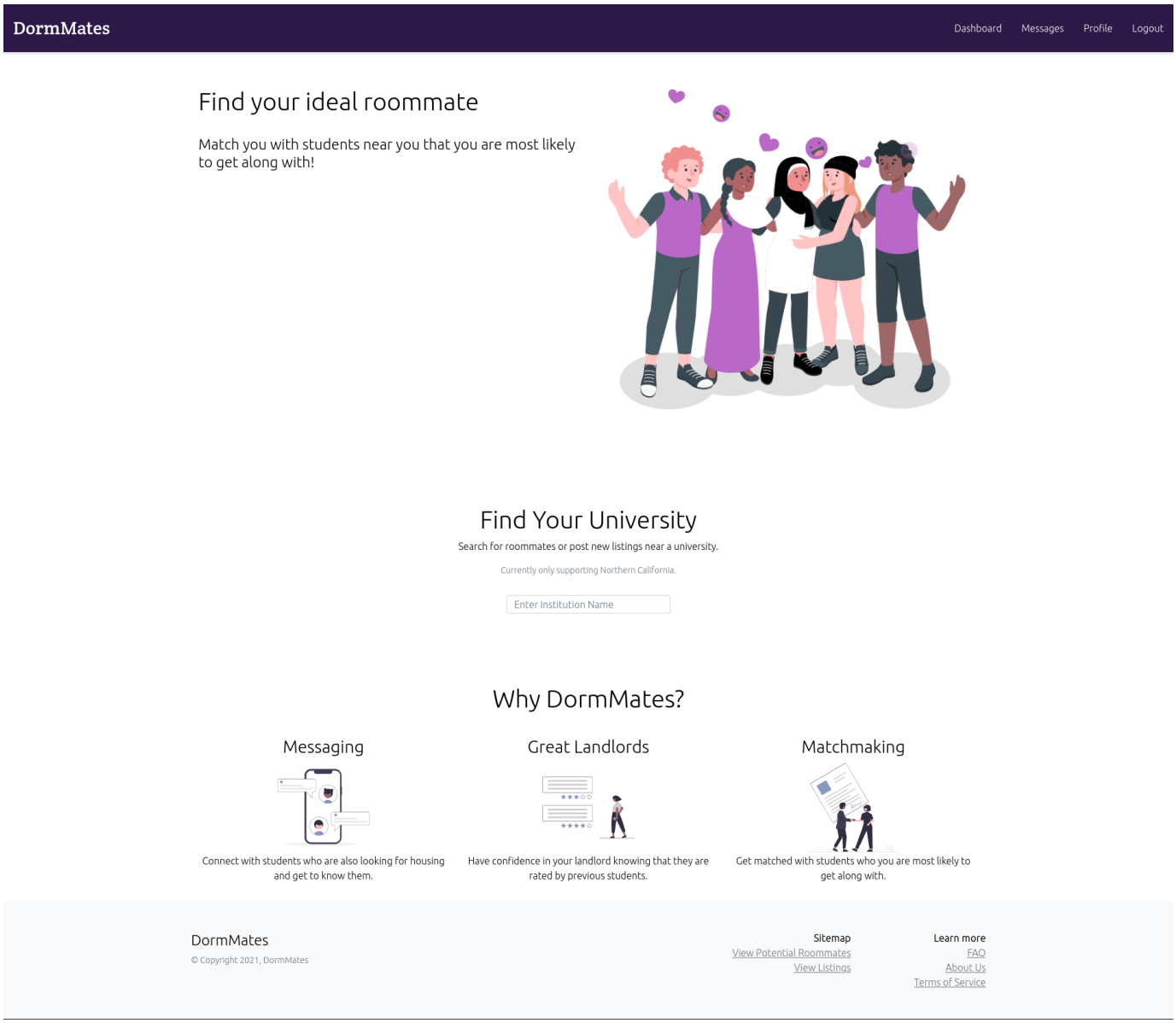
- Implemented FrontEnd Feedback to get buttons working on Safari

Ahad

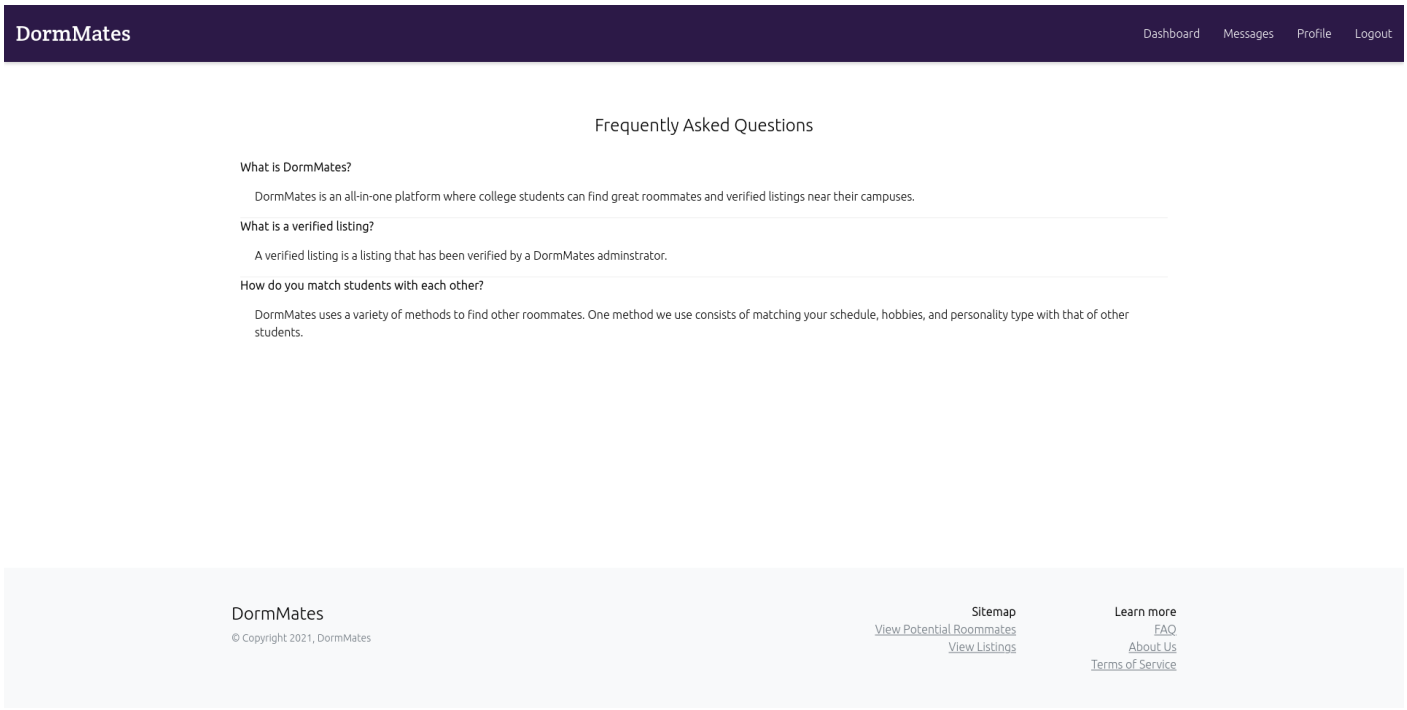
- Implementing Chat API on the backend

Product Screenshots

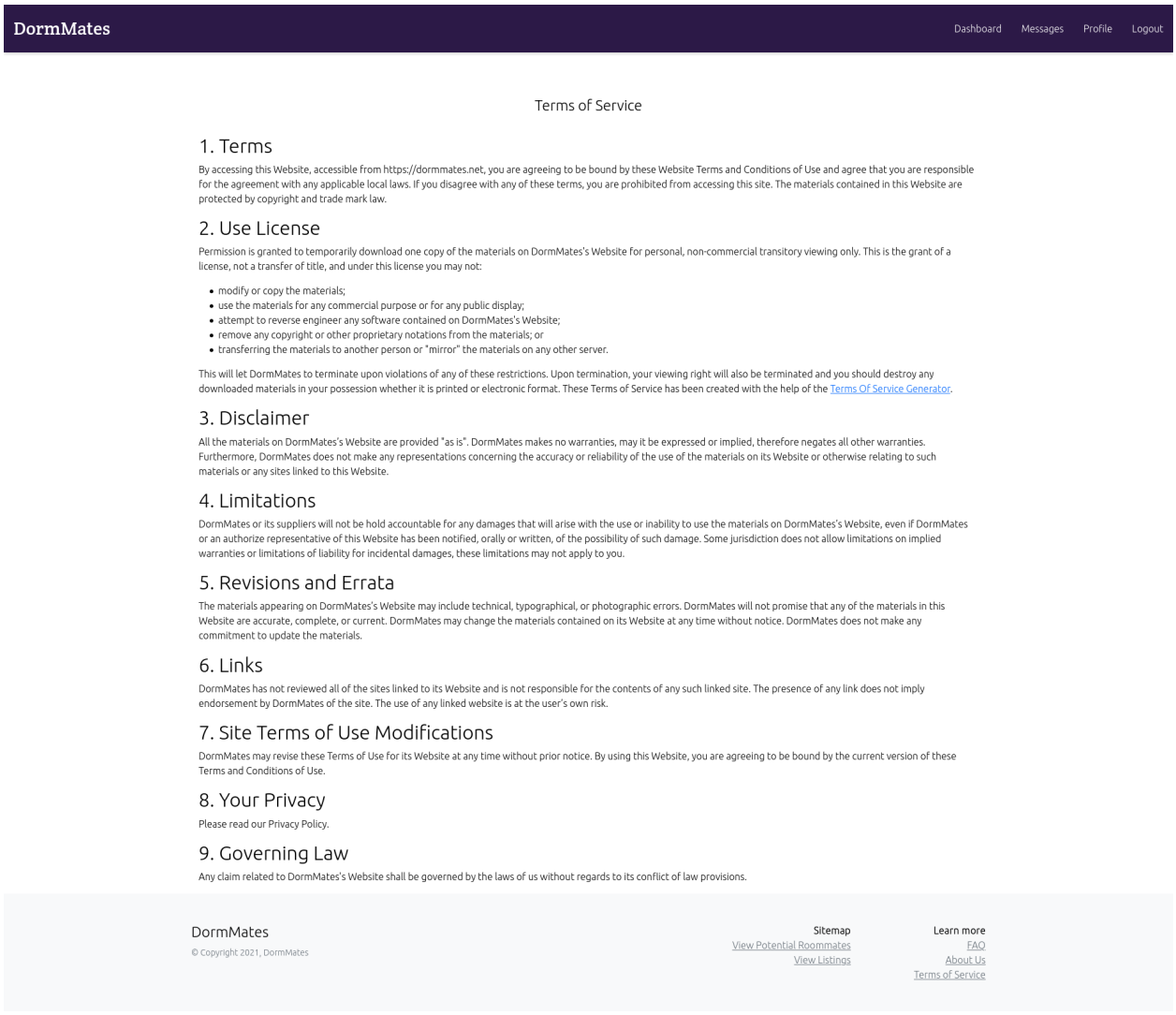
Homepage



FAQ Page


DormMates
Dashboard
Messages
Profile
Logout
Frequently Asked Questions
What is DormMates?
DormMates is an all-in-one platform where college students can find great roommates and verified listings near their campuses.
What is a verified listing?
A verified listing is a listing that has been verified by a DormMates adminstrator.
How do you match students with each other?
DormMates uses a variety of methods to find other roommates. One method we use consists of matching your schedule, hobbies, and personality type with that of other students.
DormMates
© Copyright 2021, DormMates
Sitemap
View Potential Roommates
View Listings
Learn more
FAQ
About Us
Terms of Service

Terms of Service Page



Features Page

DormMates

Dashboard Messages Profile Logout

Features

Our service provides a source of searching for roommates and housing all-in-one application. We offer various features that would make your search easier such as our roommate filter, listing filter, and messaging system.

Message anyone who shares your interests

When seeking roommates, you will have the option to direct message those who you have a compatibility with. You can get to know your future roommates beforehand and clarify your future rooming needs.

Look through landlord reviews

Having landlord reviews will give you the confidence in your housing search. Take these reviews into consideration and ensure that the landlord's expectations will match your qualifications.

Matchmake with suitable roommates

With our student questionnaire, you will be able to find students who fit your rooming criteria. Our student filtering system will also give you the option to expand your roommate search.

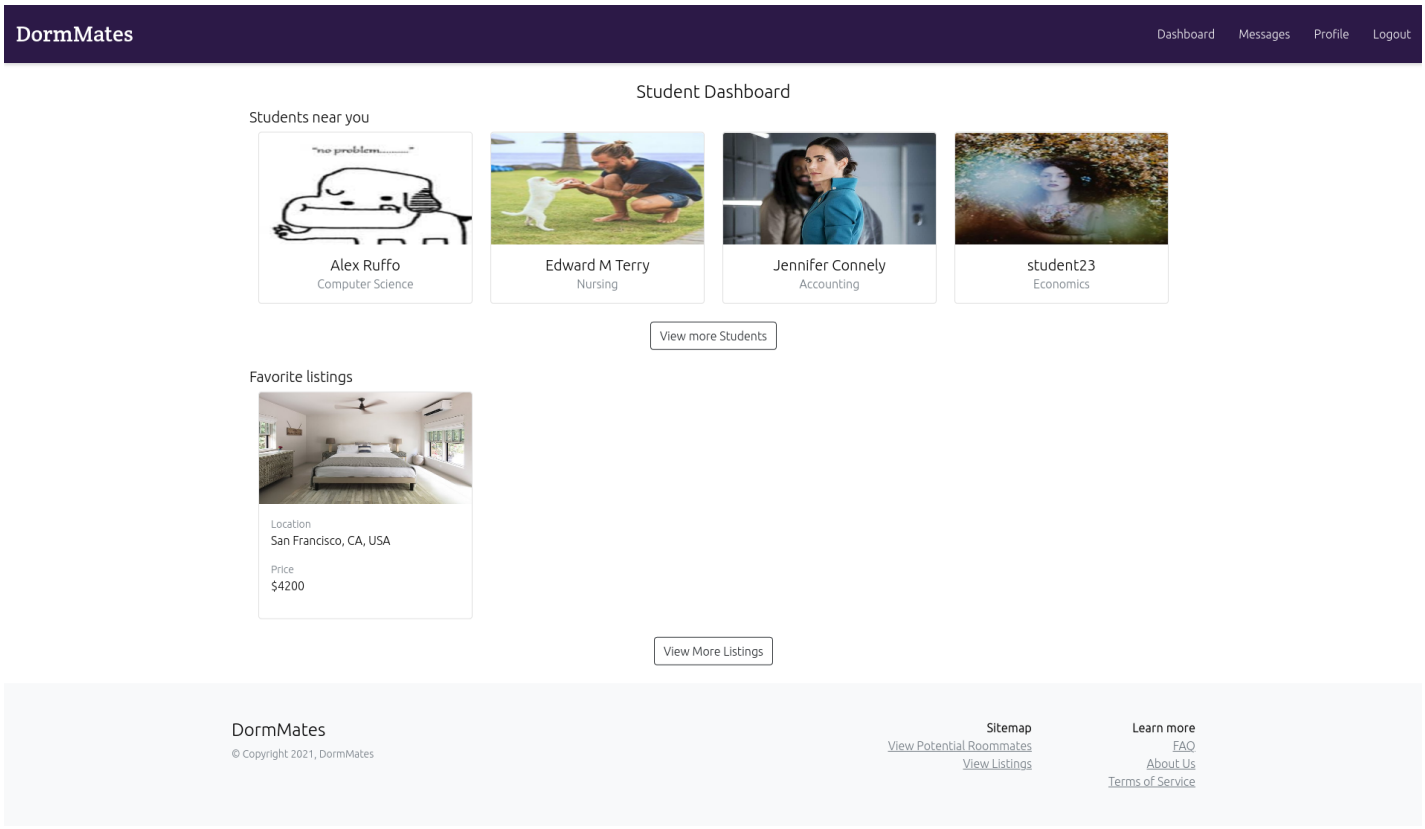
DormMates

© Copyright 2021, DormMates

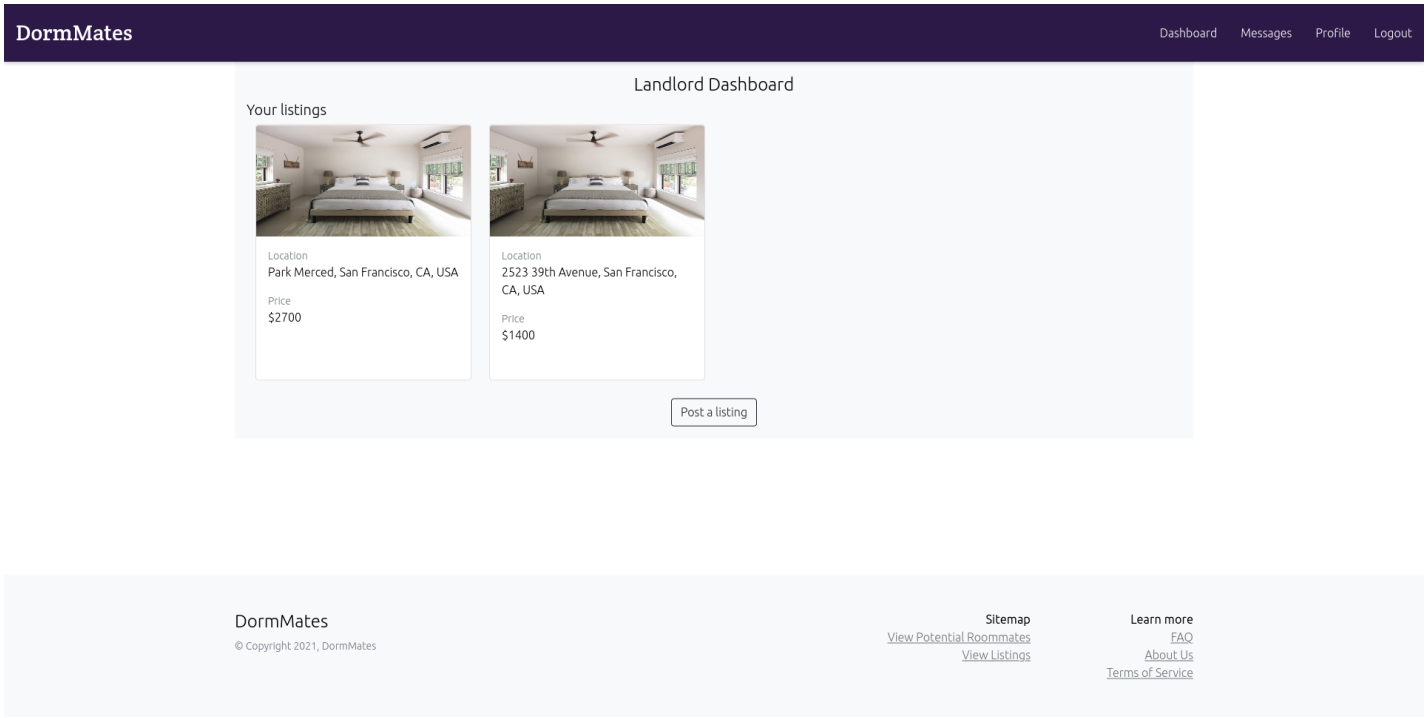
Sitemap
View Potential Roommates
View Listings

Learn more
FAQ
About Us
Terms of Service

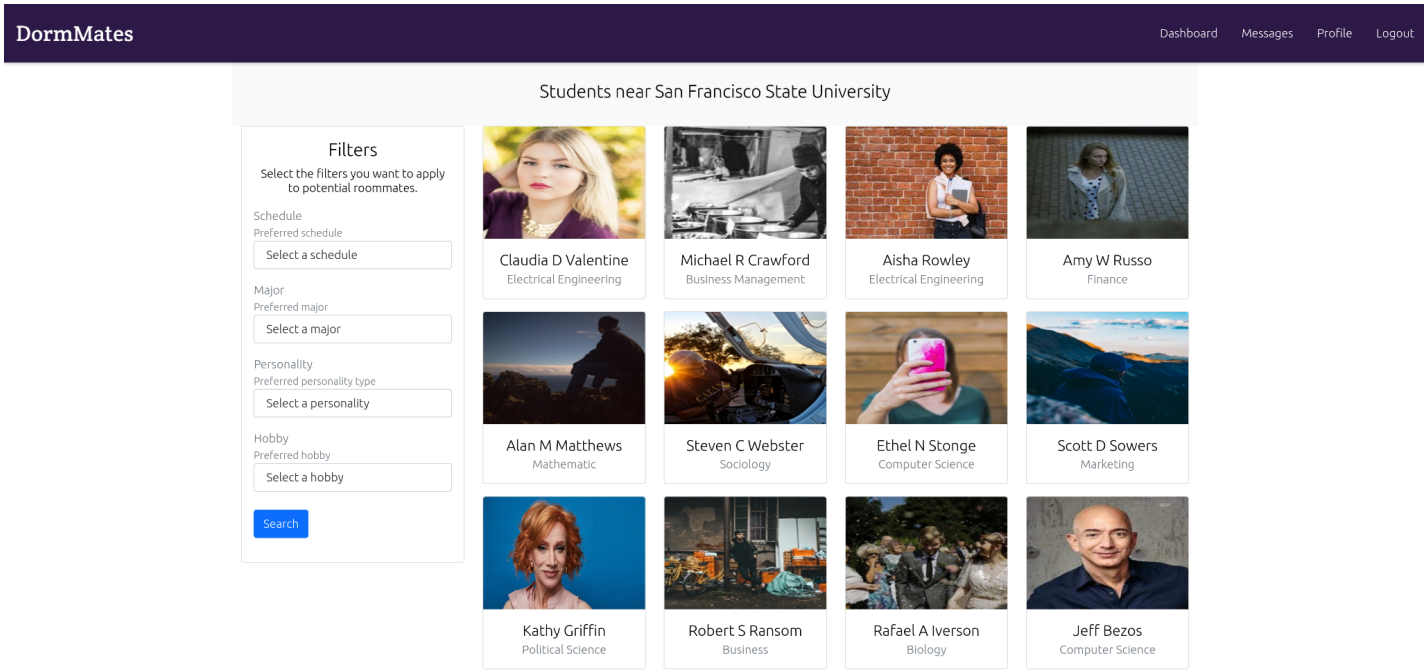
Student Dashboard Page



Landlord Dashboard Page



Roommates Search Page



Listings Page

DormMates

Dashboard Messages Profile Logout

Listings Near San Francisco State University

Filters

Select the filters you want to apply to your listings.

- [ ] Dryer
- [ ] Washer
- [ ] Whiteboard
- [ ] Furnished
- [ ] Patio
- [ ] Closet
- [ ] Parking
- [ ] Kitchen
- [ ] Bath
- [ ] Living Room
- [ ] Wifi

Apply

Location
San Francisco, CA, USA

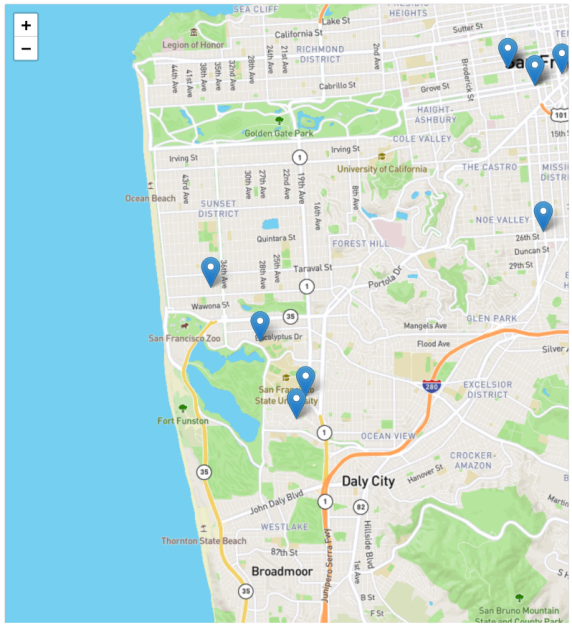
Price
$4200/month

View

Location
1066 Market St, San Francisco, CA, USA

Price
$2350/month

View



Listing Page

DormMates

Dashboard Messages Profile Logout

Listing

Description
50 Jones

Price
2350

Location
1066 Market St, San Francisco, CA, USA

Melvin P McLaughlin
Favorite
Profile

Amenities
Washer
Dryer
Wifi
Closet
Furnished
Kitchen
Whiteboard
Bath
Living Room
Patio
Parking

Availability
Available now!

DormMates
© Copyright 2021, DormMates

Sitemap
View Potential Roommates
View Listings

Learn more
FAQ
About Us
Terms of Service

Landlord Profile Page

DormMates

Dashboard Messages Profile Logout

Profile

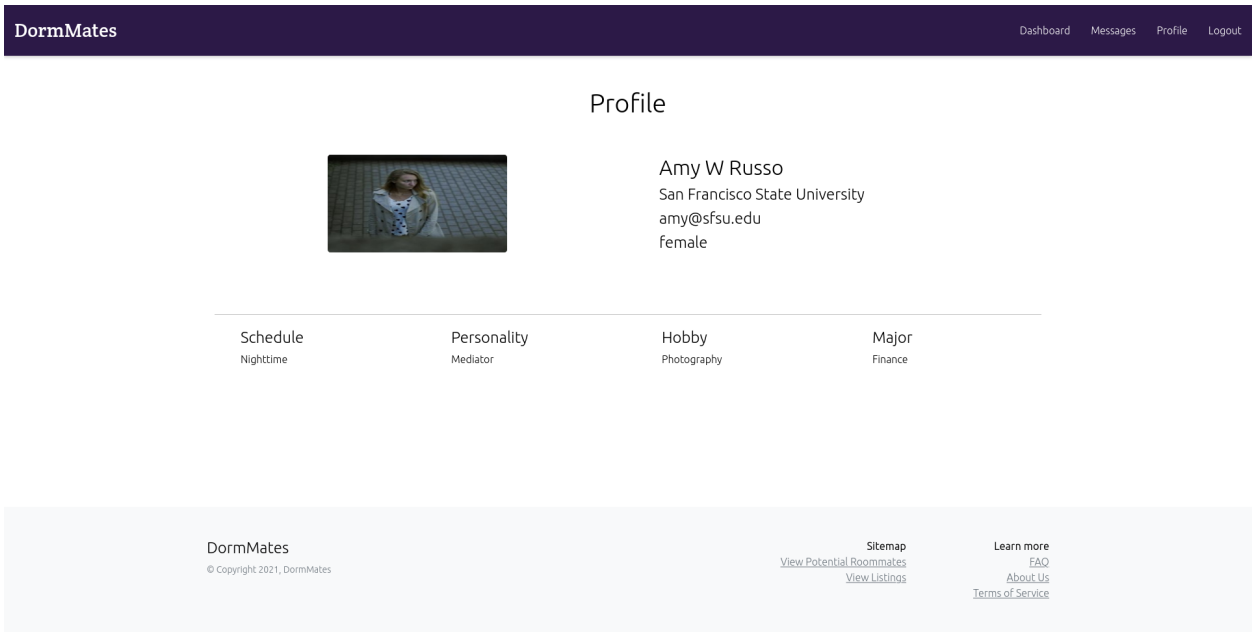
Melvin P McLaughlin
Landlord
melvin@sfsu.edu
male

DormMates
© Copyright 2021, DormMates

Sitemap
View Potential Roommates
View Listings

Learn more
FAQ
About Us
Terms of Service

Student Profile Page



DB Table Screenshots

User Table

id binary	username varchar	emailAddress varchar	password varchar	name varchar	birthdate date	lastSeen timestamp	photo mediumtext	ver...	gender ...
@***x)**'*Ey*	landlord	landlord@gmail.com	$2b$10$u82pLdGMVpGrw2VS...	landy Doe	2000-01-01	2021-08-01 05:20:59.00	...	0	Male
A*k**Hu**G*B6?*	jb	jb@sfsu.edu	$2b$10$77mKdcAuxBbzzx.0j4...	Jeff Bezos	1998-08-02	2021-07-31 01:24:02.00	...	0	male
An@o***`**U*	johndoe	jd@act-sf.org	$2b$10$su82pLdGMVpGrw2VS...	John Doe	2000-01-01	2021-08-01 05:09:18.00	...	0	M
B*ڑEN}******* ↩ o	Jcon	Jcon@sfsu.edu	$2b$10$jCp4bLkri85H7sX/Pnj...	Jennifer Connely	1995-10-31	2021-08-02 05:21:42.00	...	0	female
E ↩ K*8*w***G ↩ *1MS	andrei	ag@sfsu.edu	$2b$10$8mpyHNn0309SzsUK...	Andrei Georgescu	1998-03-23	2021-07-30 21:20:12.00	...	0	male
F*H@*I*****x^*P	johny	johny@act-sf.org	$2b$10$su82pLdGMVpGrw2VS...	johny Doe	2000-01-01	2021-08-01 05:12:58.00	...	0	Male
J;*(^3 *******,*>	qwe	qwe@sfsu.edu	$2b$10$5jTJQsgnIKj9iuAG6KM...	Alex Ruffo	1998-08-27	2021-07-30 21:23:05.00	...	0	male
K*H*****2VI7,*n	Cersei	Clannister@gmail.com	$2b$10$UQ9oatIXK4JeRMFCgr...	Cersei Lannister	1980-08-15	2021-08-02 17:11:28.00	...	0	female
K****T*	BWalters	Bwalters@sfsu.edu	$2b$10$f/YJ4xPZgXOet1Rhc6...	Barbara Walters	1952-09-15	2021-07-30 21:24:47.00	...	0	female
L*R**9***w*r**	abc	abc@gmail.com	$2b$10$OPX8iIGy0S068R4n16...	Elon Musk	1971-06-28	2021-07-31 01:29:09.00	...	0	male
MG**Ir*<*A	kg	kg@sfsu.edu	$2b$10$AKzwDVEJi6AuJF0FP...	Kathy Griffin	1998-03-23	2021-07-31 01:26:08.00	...	0	female
M}+*****:d**	janedoe	jane@act-sf.org	$2b$10$su82pLdGMVpGrw2VS...	Jane Doe	2000-01-01	2021-08-01 05:12:04.00	...	0	Male
O*]*****g***J	sj	sj@gmail.com	$2b$10$VITII2nGPN3BwAoxcT...	Steve Jobs	1998-08-02	2021-07-30 22:50:14.00	...	0	male

Student Table

id binary	userId binary	institutionID int	major varchar	personality varchar	hobby1 varchar	hobby2 varchar	schedule varchar
C*~**;ok*>**U4$p	MǴ**Ir*<*A	329	Political Science	Advocate	Food	(NULL)	Afternoon
D*:G7D**I*]v=**	A*!k**Hu**G*B6?*	329	Computer Science	Commander	Health and Fitness	(NULL)	Daytime
I*p***m****3**`	B*ڑEN}******* ↩ o	329	Accounting	Logician	Reading and Writing	(NULL)	Nighttime
J*>I*******c*ZG7	K****T*	329	Economics	Executive	Reading and Writing	(NULL)	NightTime
M*9t***p***7t**	J;*(^3 *******,*>	329	Computer Science	Logician	Food	(NULL)	Daytime
N***j*T*EC****H	E ↩ K*8*w***G ↩ *1MS	329	Computer Science	Debater	Pets	(NULL)	Afternoon

Landlord Table

id binary	userId binary	rating double	numOfRatings int
@**I*J秘***S趟	L**R***9***w*r**	0	0
B*1T**U****&u*<*	O*]*****g***J	0	0
Ĉ***ڑ**>**** ↩	K*H*****2VI7,*n	0	0
Jcq***	@*****x)**'*Ey*	0	0

Listing Table

id binary	landlordId binary	price float	location varchar	description varchar	verification tinyint	availability tinyint	listingLatitude float	listingLongitude float
A��!���R��Ʊ$$	@$J秘$S趙	4200	San Francisco, CA, USA	This property is available. Please i…	0	1	37.778	-122.431
F$	@$J秘$S趙	6000	Lakeshore Elementary School, San Francisco, C	Luxury single family home located …	0	1	37.7301	-122.486
I$$$e$%h$$ﾘﾘﾘ"I	B$1T$$U$$$$&u$<$	2700	Park Merced, San Francisco, CA, USA	Parkmerced is a unique and histori…	0	1	37.7165	-122.478
Kf8r$p>&f$$SM	B$1T$$U$$$$&u$<$	1400	2523 39th Avenue, San Francisco, CA, USA	Featuring a patio with garden view…	0	1	37.7397	-122.497
Le≠1E$$$Kp2J$J	@$J秘$S趙	2495	1250 Taylor St, San Francisco, CA, USA	Large One Bedrooms Coming Avail…	0	1	37.7942	-122.413
O8$$$$$W$$=$$	Ð$$$ڜ$$>$$$ ↩	100	455 Fair Oaks St, San Francisco, CA, USA	Quaint little dungeon in the outskirt…	0	1	37.7494	-122.423

Amenities Table

id int	listingId binary	washer tinyint	dryer tinyint	wifi tinyint	closet tinyint	furnished tinyint	kitchen tinyint	whiteboard tinyint	bath tinyint	livingroom tinyint	patio tinyint	parking tinyint
38	I$$$e$%h$$ﾘﾘﾘ"I	1	1	1	0	0	1	1	1	1	0	0
39	Kf8r$p>&f$$SM	1	1	0	0	1	1	0	1	0	0	0
40	Le≠1E$$$Kp2J$J	0	1	0	0	0	0	1	1	1	1	0
41	A$$!$$$R$$Ʊ$$	1	1	1	0	1	0	0	1	0	0	0
42	F$$	0	1	1	0	0	1	0	0	1	1	0
43	O8$$$$$W$$=$$	0	0	1	1	0	0	0	1	0	0	0

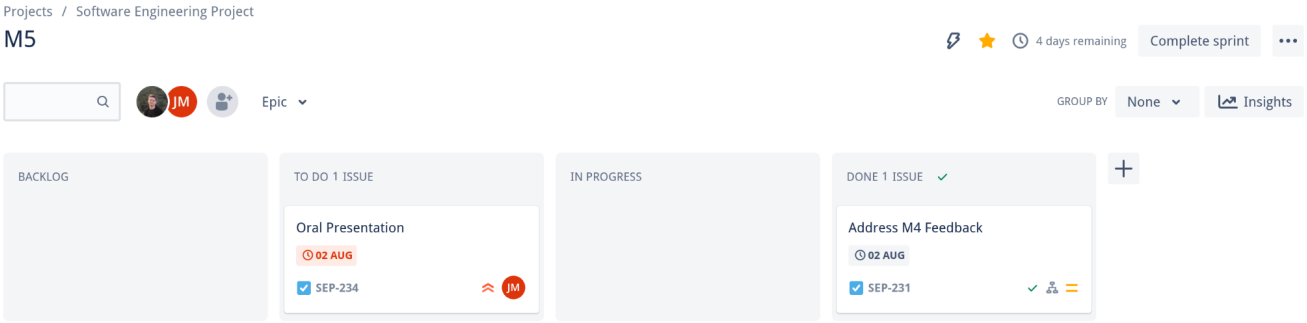
Institution Table

id int	name varchar	domain varchar	address varchar	institutionLatitude float	institutionLongitude float
270	American Conservatory Theater	act-sf.org	30 Grant Ave, San Francisco, CA 94108, USA	37.7873	-122.405
271	American River College	losrios.edu	4700 College Oak Dr, Sacramento, CA 95841, USA	38.6488	-121.347
272	Berkeley City College	berkeleycitycollege.edu	2050 Center St, Berkeley, CA 94704, USA	37.87	-122.27
273	Butte College	butte.edu	3536 Butte Campus Dr, Oroville, CA 95965, USA	39.6485	-121.645
274	Cabrillo College	cabrillo.edu	6500 Soquel Dr, Aptos, CA 95003, USA	36.9876	-121.926
275	California College of the Arts	cca.edu	5212 Broadway, Oakland, CA 94618, USA	37.8363	-122.25
276	California Maritime Academy	csum.edu	200 Maritime Academy Dr, Vallejo, CA 94590, USA	38.0689	-122.23
277	California State University Chico	csuchico.edu	400 W 1st St, Chico, CA 95929, USA	39.7288	-121.848
278	California State University East Bay	csueastbay.edu	25800 Carlos Bee Blvd, Hayward, CA 94542, USA	37.6571	-122.057
279	California State University Sacramento	csus.edu	6000 J St, Sacramento, CA 95819, USA	38.5585	-121.422
280	California State University Stanislaus	csustan.edu	1 University Cir, Turlock, CA 95382, USA	37.5252	-120.855
281	Canada College	canadacollege.edu	4200 Farm Hill Blvd, Redwood City, CA 94061, USA	37.4474	-122.265
282	Chabot College	chabotcollege.edu	25555 Hesperian Blvd, Hayward, CA 94545, USA	37.6427	-122.106
283	City College of San Francisco	ccsf.edu	50 Frida Kahlo Way, San Francisco, CA 94112, USA	37.7257	-122.451
284	Cogswell College	cogswell.edu	191 Baypointe Pkwy, San Jose, CA 95134, USA	37.414	-121.941
285	College of Alameda	peralta.edu	555 Ralph Appezzato Memorial Pkwy, Alameda, CA 94501, USA	37.7823	-122.279
286	College of Marin	marin.edu	835 College Ave, Kentfield, CA 94904, USA	37.9558	-122.55
287	College of San Mateo	collegeofsanmateo.edu	1700 W Hillsdale Blvd, San Mateo, CA 94402, USA	37.5357	-122.335
288	College of the Redwoods	redwoods.edu	7351 Tompkins Hill Rd, Eureka, CA 95501, USA	40.6984	-124.195
289	College of the Siskiyous	siskiyous.edu	800 College Ave, Weed, CA 96094, USA	41.4122	-122.39
290	Columbia College	gocolumbia.edu	11600 Columbia College Dr, Sonora, CA 95370, USA	38.0318	-120.389
291	Contra Costa College	contracosta.edu	2600 Mission Bell Dr, San Pablo, CA 94806, USA	37.9689	-122.34
292	Cosumnes River College	crc.losrios.edu	8401 Center Pkwy, Sacramento, CA 95823, USA	38.4541	-121.423
293	De Anza College	deanza.edu	21250 Stevens Creek Blvd, Cupertino, CA 95014, USA	37.3209	-122.043
294	Dell Arte International School of Physical Theatre	dellarte.com	131 H St, Blue Lake, CA 95525, USA	40.8804	-123.99
295	Diablo Valley College	dvc.edu	321 Golf Club Rd, Pleasant Hill, CA 94523, USA	37.9687	-122.071
296	Dominican University of California	dominican.edu	50 Acacia Ave, San Rafael, CA 94901, USA	37.98	-122.513
297	Epic Bible College	EPIC.edu	4330 Auburn Blvd, Sacramento, CA 95841, USA	38.6453	-121.363
298	Evergreen Valley College	evc.edu	3095 Yerba Buena Rd, San Jose, CA 95135, USA	37.3011	-121.765

Task Management System Screenshots

We utilized Jira for our task management and in total we had over 170 different issues. For each milestone we had a sprint and each sprint consisted of high level tasks that contained several subtasks within them required for completing the high level task.

Screenshot of Sprint in Jira



Screenshots of Some High-Level Tasks and Subtasks

Type	Key	Summary	Assignee	Reporter	P	Status	Created
☑	SEP-213	Current State of Backend	Andrei Georgescu	Andrei Georgescu	⏫	DONE	Jul 24, 2021
☑	SEP-211	Frontend team implements feedback from the group	Meeka C.	Andrei Georgescu	↑	DONE	Jul 24, 2021
☑	SEP-208	QA Test Plan	Meeka C.	Andrei Georgescu	↑	DONE	Jul 24, 2021
☑	SEP-207	Usability Test Plan	Jonathan McGrath	Andrei Georgescu	↑	DONE	Jul 24, 2021
☑	SEP-199	Give feedback to the frontend team	Unassigned	Andrei Georgescu	=	DONE	Jul 23, 2021
☑	SEP-183	Post Listing, Roommates, and Listings Page	mirzahihamid	Andrei Georgescu	↑	DONE	Jul 13, 2021
☑	SEP-182	Dashboard and Questionnaire Pages	Jimmy Yeung	Andrei Georgescu	↑	DONE	Jul 13, 2021
☑	SEP-181	Profile, Messages, and Listing Pages	Meeka C.	Andrei Georgescu	↑	DONE	Jul 13, 2021
☑	SEP-155	Group & Chat API	Ahad Zafar	Andrei Georgescu	=	DONE	Jul 13, 2021
☑	SEP-152	Listing & Lease API	Jonathan McGrath	Andrei Georgescu	=	DONE	Jul 13, 2021
☑	SEP-151	Search & Favorites API	Alex Ruffo	Andrei Georgescu	=	DONE	Jul 13, 2021
☑	SEP-149	Create an outline for frontend wireframes.	Meeka C.	Andrei Georgescu	↑	DONE	Jul 13, 2021
☑	SEP-148	Address feedback for the "Prioritized Functional Requirements" task.	Andrei Georgescu	Andrei Georgescu	↑	DONE	Jul 12, 2021

Type	Key	Summary	Assignee	Reporter	P	Status	Created ↓
☑	SEP-112	Setup and configure express-sessions	Andrei Georgescu	Andrei Georgescu	=	DONE	Jul 2, 2021
☑	SEP-111	Insert SQL data into production database	Ahad Zafar	Andrei Georgescu	=	DONE	Jul 2, 2021
☑	SEP-110	Convert CSV to SQL	Ahad Zafar	Andrei Georgescu	=	DONE	Jul 2, 2021
☑	SEP-109	Create a CSV of all colleges in Northern California with their Name, Location, and Domain	Ahad Zafar	Andrei Georgescu	=	DONE	Jul 2, 2021
☑	SEP-108	Compile a list of all colleges in Northern California	Ahad Zafar	Andrei Georgescu	=	DONE	Jul 2, 2021
☑	SEP-107	Create registration scripts	mirzahihamid	Andrei Georgescu	=	DONE	Jul 2, 2021
☑	SEP-106	Create registration pug views	mirzahihamid	Andrei Georgescu	=	DONE	Jul 2, 2021
☑	SEP-105	Create registration wireframe(s)	mirzahihamid	Andrei Georgescu	=	DONE	Jul 2, 2021
☑	SEP-104	Create scripts to be used by the homepage	Meeka C.	Andrei Georgescu	=	DONE	Jul 2, 2021
☑	SEP-103	Create CSS styles	Meeka C.	Andrei Georgescu	=	DONE	Jul 2, 2021
☑	SEP-102	Create homepage pug view	Meeka C.	Andrei Georgescu	=	DONE	Jul 2, 2021
☑	SEP-101	Create homepage wireframe(s)	Meeka C.	Andrei Georgescu	=	DONE	Jul 2, 2021
☑	SEP-97	Registration Vertical Prototype	mirzahihamid	Andrei Georgescu	=	DONE	Jul 2, 2021

Screenshot of Detailed View of a High-Level Task

Projects / Software Engineering... / Add epic / SEP-46

2

High Level Database Architecture and Organization

Attach | Add a child issue | Link issue

Description

Complete all Database tasks and deploy the approved database schema to our AWS RDS server.

Child issues

Order by

100% Done

SEP-85	Define all business rules	JM	DONE
SEP-86	Create a list of entities, attributes, and relationships	JM	DONE
SEP-87	Create an ERD diagram for business rules.	JM	DONE
SEP-88	Create a MySQL database model based on the ERD	JM	DONE
SEP-89	Document why we chose MySQL, how we will store me...	JM	DONE

Activity

Show: All | Comments | History — Newest first

Add a comment...

Pro tip: press M to comment

Done — Done

Pinned fields

Priority	High
Due date	Jul 03, 2021
Assignee	Jonathan McGrath

Details

Labels	backend
Sprint	None +1
Reporter	Andrei Georgescu

More fields

Story point estimate	None

Created June 22, 2021, 5:25 PM
Updated July 7, 2021, 10:41 AM
Resolved July 7, 2021, 10:41 AM

Configure

Team Member Contributions

Member	Contribution Score	Explanation
Andrei	10	- Hosted team meetings to discuss the product - Delegated tasks to different team members - Worked with team members to design and implement key features
Jonathan	10	- Completed all assigned tasks - Communicated very frequently with the team - Attended all team meetings - Participated in all team meetings
Alexandre	10	- Completed all assigned tasks - Communicated very frequently with the team - Attended all team meetings - Participated in all team meetings
Meeka	10	- Completed all assigned tasks - Communicated very frequently with the team - Attended all team meetings - Participated in all team meetings
Jimmy	9	- Completed all assigned tasks - Communicated frequently with the team - Attended all team meetings - Participated in all team meetings
Sayed	6	- Completed most assigned tasks - Communicated somewhat frequently with the team - Attended most meetings - Participated in most team meetings.
Ahad	4	- Attended all team meetings - Participated in all team meetings - Did not communicate frequently with the team - Completed assigned tasks last minute or late which resulted in missed P1 functionality

Post Analysis

Overview

For the most part, every team member worked together and completed most of the tasks that were assigned to them. While there were some rough patches that the team went through during a few milestones, in the end we came out with a product that we should all be proud to put on our resumes.

Main Challenges

There were a few main challenges that we experienced throughout this project. One of them was a lack of commitment and another was team management. When it comes to commitment, some group members did not treat this project like an actual industry job and this resulted in unnecessary stress being put on the rest of the team. In addition to a lack of commitment, communication is also something that was missing as a result of the lack of commitment. In terms of team management, the team lead could have done a much better job at managing the team and addressing these challenges as they came up rather than giving gentle warnings and waiting for improvement. In some instances, the team lead was too gentle with the warnings and was not so strict on deadlines. Looking back now, the team lead could have also done a better job of communicating the challenges to individual members and the CTO.

Addressing These Challenges

Addressing the lack of commitment is something that students must do on their own. At the end of the day, it all comes down to how much you care about the grade you get in the class. I anticipate that this will not be something students struggle with when they are actually working in industry because money is a much greater motivator than a GPA. In terms of team management, one thing that I would do differently is hold people more accountable for their

tasks. By this I mean that missing one or two deadlines would be allowed, but everything after that should have been reported to the CTO. In addition to this, any tasks that do not get completed should not always just get taken over by someone else on the team. Giving people this opportunity is something I struggled with and I realize that it placed a great amount of stress on the team and caused tension between members in the team. Something that I think would help with this is giving people more ownership over certain aspects of the project.